

モータドライブ基板 TypeS Ver.3 アナログセンサ基板 TypeS Ver.2 プログラム解説マニュアル (R8C/38A 版)

2013年度から、RY_R8C38ボードに搭載されているマイコンがR8C/38AからR8C/38Cに変更されました。R8C/38AマイコンとR8C/38Cマイコンは、機能的にほぼ互換で、マイコンカーで使う範囲においてはプログラムの変更はほとんどありません。よって、本マニュアルではマイコンの名称を『R8C/38A』で統一します。

本マニュアルで 使用している基板内容	<ul style="list-style-type: none">●モータドライブ基板 TypeS Ver.3●アナログセンサ基板 TypeS Ver.2 (アナログセンサ基板 TypeS も使用可能です)
本基板の 対象マイコンボード	RY_R8C38 ボード

第2.16版

2015.04.20

ジャパンマイコンカーラリー実行委員会
株式会社日立ドキュメントソリューションズ

注意事項 (rev.6.0J)

著作権

- ・本マニュアルに関する著作権はジャパンマイコンカーラリー実行委員会に帰属します。
- ・本マニュアルは著作権法および、国際著作権条約により保護されています。

禁止事項

ユーザーは以下の内容を行うことはできません。

- ・第三者に対して、本マニュアルを販売、販売を目的とした宣伝、使用、営業、複製などを行うこと
- ・第三者に対して、本マニュアルの使用権を譲渡または再承諾すること
- ・本マニュアルの一部または全部を改変、除去すること
- ・本マニュアルを無許可で翻訳すること
- ・本マニュアルの内容を使用しての、人命や人体に危害を及ぼす恐れのある用途での使用

転載、複製

本マニュアルの転載、複製については、文書によるジャパンマイコンカーラリー実行委員会の事前の承諾が必要です。

責任の制限

本マニュアルに記載した情報は、正確を期すため、慎重に制作したのですが万一本マニュアルの記述誤りに起因する損害が生じた場合でも、ジャパンマイコンカーラリー実行委員会はその責任を負いません。

その他

- ・本マニュアルに記載の情報は本マニュアル発行時点のものであり、ジャパンマイコンカーラリー実行委員会は、予告なしに、本マニュアルに記載した情報または仕様を変更することがあります。製作に当たりましては、最新の内容を確認いただきますようお願いいたします。
- ・すべての商標および登録商標は、それぞれの所有者に帰属します。

連絡先

株式会社 日立ドキュメントソリューションズ

〒135-0016 東京都江東区東陽六丁目 3 番 2 号 イースト 21 タワー

E-mail:himdx.m-carrally.dd@hitachi.com

目次

1. 仕様.....	1
1.1 構成.....	1
1.2 参照.....	1
2. アナログセンサ基板 TypeS Ver.2.....	2
2.1 仕様.....	2
2.2 外観.....	3
2.3 デジタルセンサとアナログセンサ.....	4
2.4 アナログセンサで使用する素子.....	5
2.5 アナログセンサの動作原理.....	6
2.6 マイコンで電圧を取り込む.....	7
2.7 コースの状態を取り込む.....	7
3. モータドライブ基板 TypeS Ver.3.....	9
3.1 仕様.....	9
3.2 ラジコンサーボと自作サーボ.....	10
3.3 モータドライブ回路.....	11
3.3.1 モータの回し方(電圧と動作の関係).....	11
3.3.2 Hブリッジ回路.....	12
3.3.3 スイッチをFETにする.....	12
3.3.4 スピード制御.....	13
3.3.5 正転とブレーキの切り替え時にショートしてしまう.....	14
3.3.6 PチャンネルとNチャンネルの短絡防止回路.....	15
3.3.7 PチャンネルとNチャンネルの短絡防止回路.....	17
3.3.8 実際の回路.....	18
3.3.9 正転、ブレーキ時の動作.....	19
3.3.10 逆転、ブレーキ時の動作.....	20
3.3.11 正転、フリー時の動作.....	21
3.3.12 マイコンのポート割り振り.....	22
3.4 ロータリエンコーダ信号入力回路.....	23
3.5 ブザー回路.....	23
3.6 ボリューム信号入力回路.....	24
3.7 信号入力回路.....	24
3.8 LED、ディップスイッチ回路.....	25
3.8.1 LEDを点灯させるとき.....	25
3.8.2 ディップスイッチの状態を読み込むとき.....	26
3.9 プッシュスイッチ回路.....	26
4. 説明用マイコンカーの仕様.....	27
4.1 寸法.....	27
4.2 サーボ機構の自作.....	28
4.3 ブロック図.....	29
4.4 R8C/38A マイコンで使用する内蔵周辺機能.....	29
5. ワークスペース「anaservo_ver3_38a」.....	30
5.1 ワークスペースのインストール.....	30

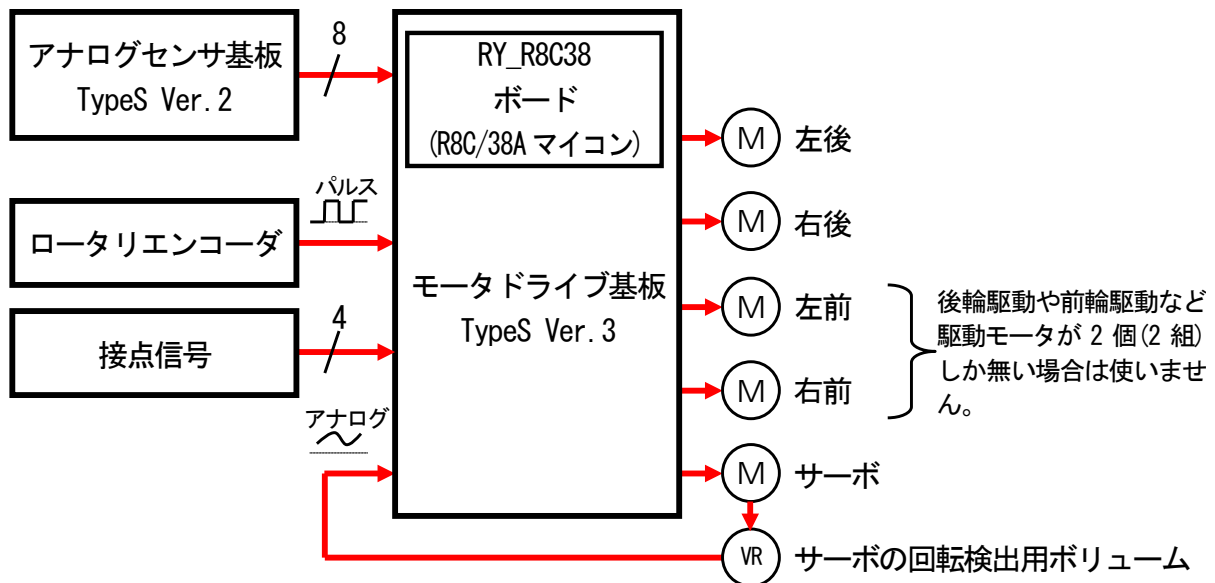
5.2 プロジェクト.....	32
5.3 プロジェクトの構成.....	33
6. マイコンカー走行プログラムの解説.....	34
6.1 プログラムリスト「anaservo_ver3_38a.c」.....	34
6.2 プログラムの解説.....	45
6.2.1 シンボル定義.....	45
6.2.2 変数の定義.....	46
6.2.3 内輪差値計算用の配列追加.....	47
6.2.4 クロックの選択.....	49
6.2.5 ポートの入出力設定.....	50
6.2.6 タイマ RB の設定.....	52
6.2.7 A/D コンバータの設定.....	54
6.2.8 タイマ RG の設定.....	56
6.2.9 タイマ RC の設定.....	58
6.2.10 タイマ RD の設定.....	68
6.2.11 タイマ RB の 1ms ごとの割り込みプログラム.....	78
6.2.12 アナログセンサ基板 TypeS Ver.2 のデジタルセンサ値読み込み.....	81
6.2.13 アナログセンサ基板 TypeS Ver.2 の中心デジタルセンサ読み込み.....	82
6.2.14 アナログセンサ基板 TypeS Ver.2 のスタートバー検出センサ読み込み.....	83
6.2.15 RY_R8C38 ボード上のディップスイッチ値読み込み.....	84
6.2.16 モータドライブ基板 TypeS Ver.3 上のディップスイッチ値読み込み.....	85
6.2.17 モータドライブ基板 TypeS Ver.3 上のプッシュスイッチ値読み込み.....	86
6.2.18 モータドライブ基板 TypeS Ver.3 の CN6 の状態読み込み.....	87
6.2.19 モータドライブ基板 TypeS Ver.3 の LED 制御.....	88
6.2.20 後輪の速度制御.....	89
6.2.21 後輪の速度制御 2 ディップスイッチには関係しない motor 関数.....	91
6.2.22 前輪の速度制御.....	92
6.2.23 前輪の速度制御 2 ディップスイッチには関係しない motor 関数.....	94
6.2.24 後モータ停止動作(ブレーキ、フリー)設定.....	95
6.2.25 前モータ停止動作(ブレーキ、フリー)設定.....	95
6.2.26 サーボモータの速度制御.....	96
6.2.27 クロスラインの検出処理.....	97
6.2.28 サーボモータ角度の取得.....	98
6.2.29 アナログセンサ値の取得.....	99
6.2.30 サーボモータ制御.....	101
6.2.31 内輪 PWM 値計算.....	104
6.2.32 main 関数-初期化.....	105
6.2.33 パターン処理.....	106
6.2.34 パターン 0:スタート待ち.....	106
6.2.35 パターン 1:スタートバー開待ち.....	107
6.2.36 パターン 11:通常トレース.....	108
6.2.37 パターン 21:クロスライン検出処理.....	109
6.2.38 パターン 22:クロスライン後のトレース、直角検出処理.....	110
6.2.39 パターン 31:右クランク処理.....	112
6.2.40 パターン 32:右クランク処理後、少し時間がたつまで待つ.....	113
6.2.41 パターン 41:左クランク処理.....	113
6.2.42 パターン 42:左クランク処理後、少し時間がたつまで待つ.....	113
6.3 ブザー制御プログラムの解説.....	114
6.3.1 ブザー関連変数の初期化.....	114

6.3.2	ブザーの出力パターンセット	115
6.3.3	ブザー処理	115
7.	調整のポイント	116
7.1	サーボモータの回転方向	116
7.2	ボリュームの調整	116
7.3	角度を測っておく	116
7.4	プログラムの調整のポイント	117
8.4	4 輪の回転数計算	119
8.1	センターピボット方式 4 輪の回転数計算	119
8.2	アッカーマン方式 4 輪の回転数計算	121
9.	自作サーボモータの角度指定	125
9.1	PD 制御	125
9.2	プログラム	125
9.2.1	グローバル変数の追加	125
9.2.2	関数の追加	126
9.2.3	割り込みプログラムの追加	126
9.2.4	使い方	127
10.	参考文献	128

1. 仕様

1.1 構成

本マニュアルは、下記構成のマイコンカーを対象に説明しています。



- モータドライブ基板 TypeS Ver.3 使用 (RY_R8C38 ボード使用)
- アナログセンサ基板 TypeS Ver.2 使用
- サーボモータ (ステアリング機構) の回転検出ボリューム搭載済み
- ロータリエンコーダ搭載済み (1 回転あたりのパルス数と距離はプログラム内で設定します)

1.2 参照

それぞれの基板、機器の詳細な説明は下表のマニュアルを参照してください。

基板、機器名	キット、製作についてのマニュアル	プログラムについてのマニュアル
モータドライブ基板 TypeS Ver.3	モータドライブ基板 TypeS Ver.3 製作マニュアル	本マニュアル
アナログセンサ基板 TypeS Ver.2	アナログセンサ基板 TypeS Ver.2 製作マニュアル (R8C/38A 版)	本マニュアル
ロータリエンコーダ	ロータリエンコーダ Ver.2 製作マニュアル (R8C/38A 版)	ロータリエンコーダ kit07_38a プログラム解説マニュアル (R8C/38A 版)

2. アナログセンサ基板 TypeS Ver.2

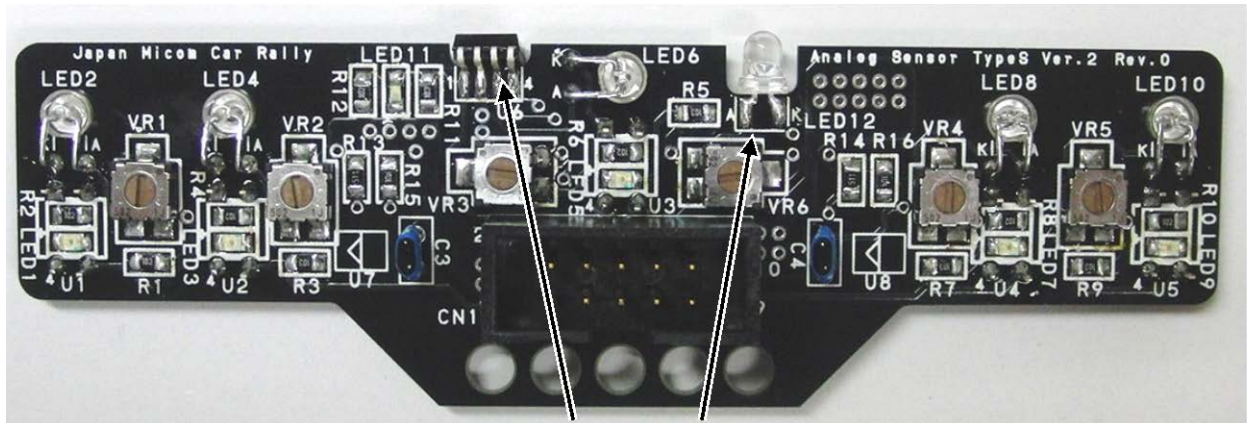
2.1 仕様

下記に、アナログセンサ基板 TypeS Ver.2 の仕様を示します。

名称	アナログセンサ基板 TypeS Ver.2
販売開始時期	2012 月 2 月
コースを見るデジタルセンサの個数	5 個
コースを見るアナログセンサの個数	2 個
スタートバーを見るセンサの個数	1 個
デジタルセンサの信号反転方法	プログラムで反転
電圧	DC5.0V±10%
重量 (基板のみ)	約 3g
重量 (完成品の実測)	約 8g ※重量は、リード線の長さや半田の量で変わります
レジスト (基板色)	黒色
基板寸法	W94×D28×厚さ 1.0mm
部品実装時の寸法 (実測)	最大 W94×D28×H13mm

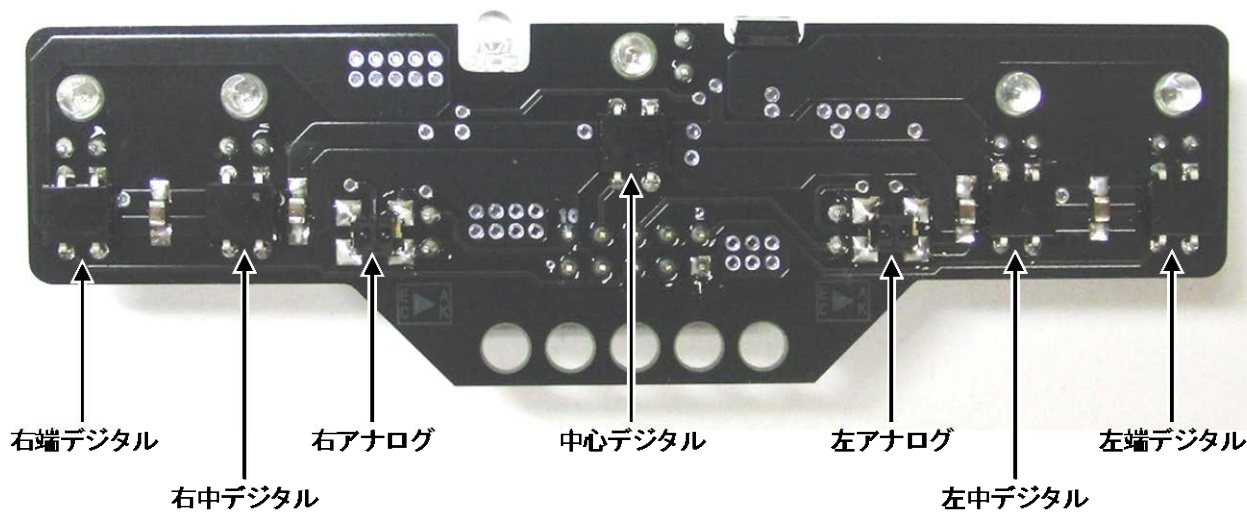
2.2 外観

部品面



スタートバー検出センサ(左が受光側、右が発光側)

半田面



※半田面は、裏から見ているため左右が逆になります。

詳しくは、アナログセンサ基板 TypeS Ver.2 製作マニュアルを参照してください。

2. アナログセンサ基板 TypeS Ver.2

2.3 デジタルセンサとアナログセンサ

マイコンカーで使用する場合の、デジタルセンサとアナログセンサの特徴を下記に示します。

項目	デジタルセンサ	アナログセンサ
回路例	<p style="text-align: center;">センサのピン振り</p> <p>1: 赤外 LED のカソード 2: +電源 3: 出力 4: GND</p>	<p style="text-align: center;">センサのピン振り</p> <p>1: エミッタ 2: コレクタ 3: カソード 4: アノード</p>
センサ出力	センサ下部が白色(灰色)か黒色かの判断	センサ下部が、白色か灰色か黒色か、さらに黒に近い灰色か、白に近い灰色かなど、細かく検出可能
外乱	強い、S7136 は特に強い	非常に弱い
コースとの間隔	2mm~10mm、幅が広い	約 2~4mm 常に一定にする必要がある
ポート	センサからの信号はデジタル出力なので、マイコンのどのポートでも入力可能	センサからの信号はアナログ出力なので、マイコンのアナログ入力端子のみで入力可能

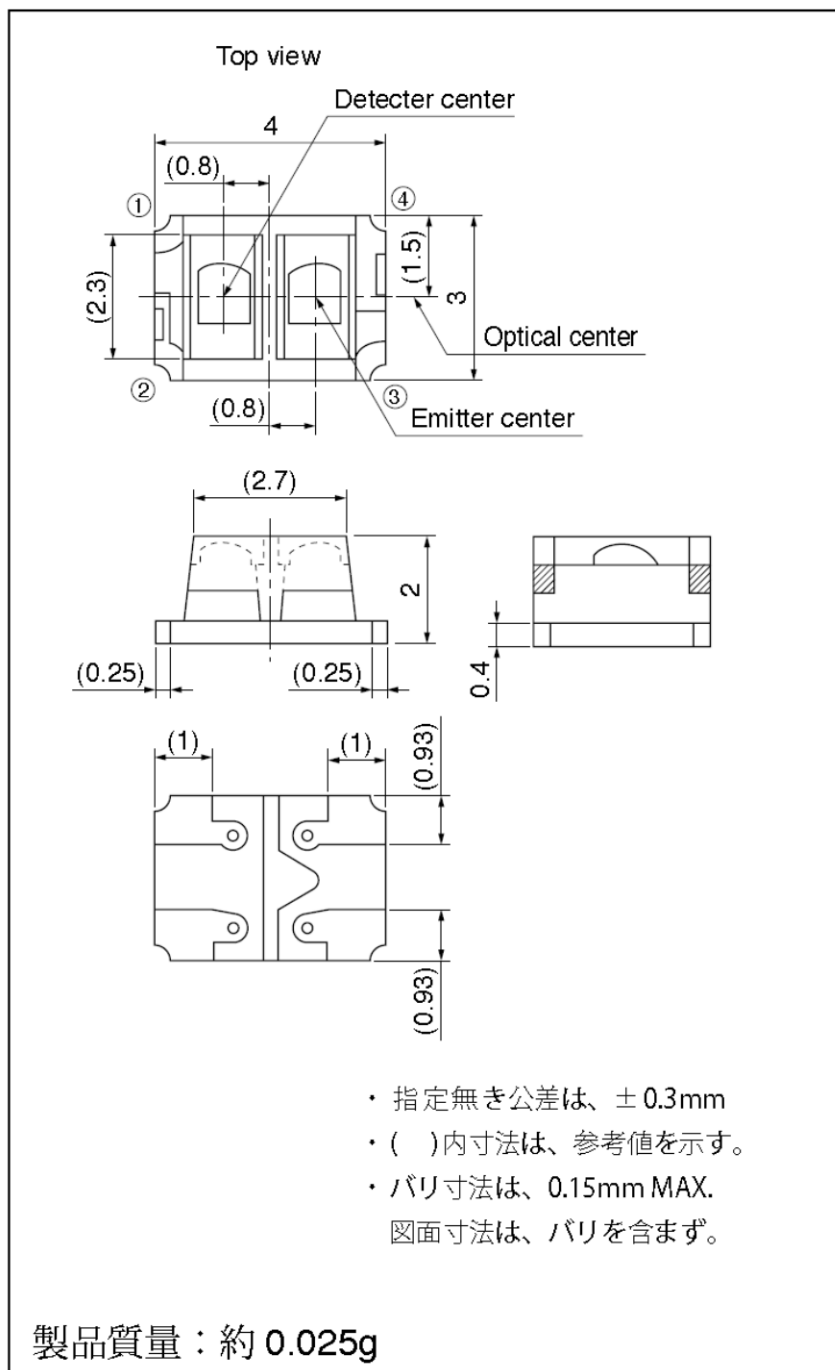
アナログセンサはデジタルセンサに比べ、外乱に弱いですがコースの状態を細かく知ることができます。この情報をうまく使えばステアリング制御を非常に細かくできるため、ライントレースを非常に滑らかに行うことができます。

2.4 アナログセンサで使用する素子

アナログセンサ基板 TypeS Ver.2 では、アナログセンサとしてシャープ(株)製の「GP2S700」というフォトインタラプタを使用しています。外形を下記に示します。

■ 外形寸法図

(単位：mm)

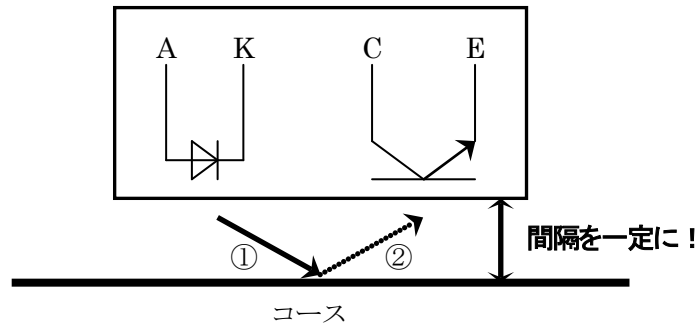


※GP2S700 のデータシートより抜粋

4.0×3.0mm 角の中に発光部である赤外 LED と、受光部であるフォトランジスタが内蔵されており、非常にコンパクトです。この素子をアナログセンサ基板 TypeS Ver.2 の半田面に取り付けます。

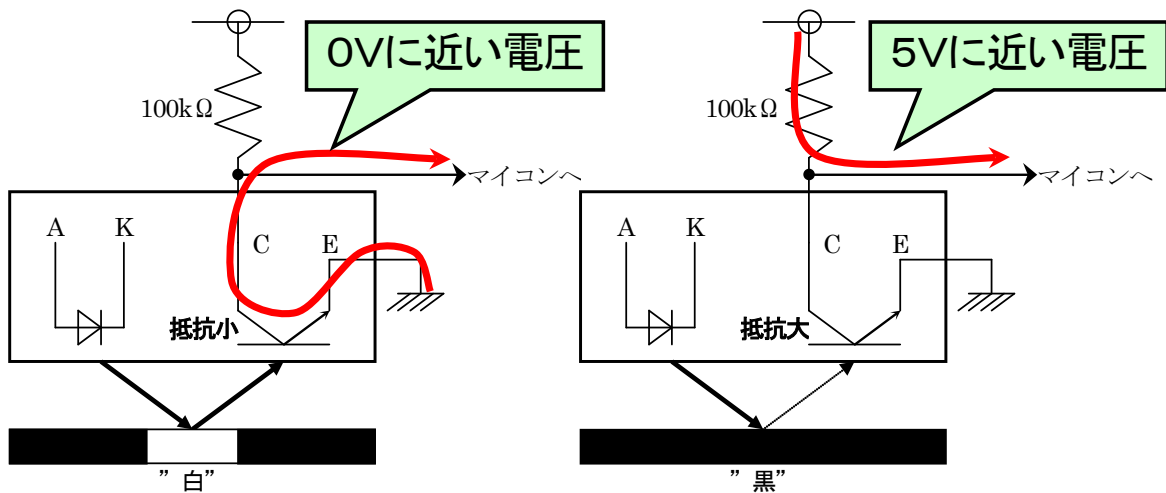
2.5 アナログセンサの動作原理

アナログセンサは、赤外 LED とフォトトランジスタの 1 組で構成されています。赤外 LED から出力される光をコースに当てて、その反射光をフォトトランジスタで受けます(下図)。このとき、白色は光を反射、黒色は吸収することを利用します。センサ下部が白色なら、赤外 LED から出た光は多く反射して、フォトトランジスタに届きます。黒色なら反射が少ないのでフォトトランジスタにあまり届きません。灰色はその中間です。



- ① 赤外 LED 側から赤外線を出します。
- ② コースに反射した光をフォトトランジスタで受けます。

コースの色と、フォトトランジスタの出力電圧の関係を下記に示します。



センサ下部が白色なら、フォトトランジスタに多くの光が届くのでエミッターコレクタ間の抵抗が少なり、マイコンへは低い電圧が出力されます。

センサ下部が黒色なら、フォトトランジスタに光りがあまり届かないのでエミッターコレクタ間の抵抗が大きくなり、コレクタに接続されているプルアップ抵抗を通してマイコンへは高位電圧が出力されます。

灰色は、その中間です。

このように、電圧の違いで、黒色、灰色、白色を判断することができます。

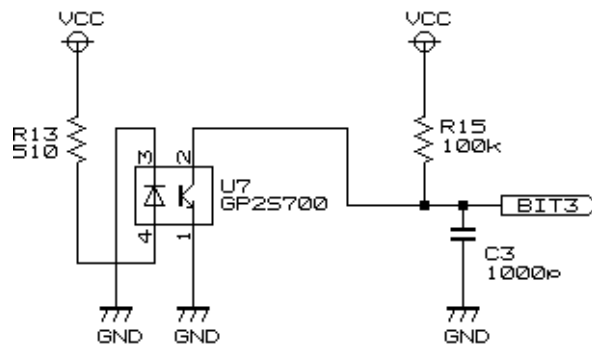
2.6 マイコンで電圧を取り込む

R8C/38A マイコンには A/D 変換器が内蔵されていて、端子に入力されている電圧を知ることができます。ポイントを、下記に示します。

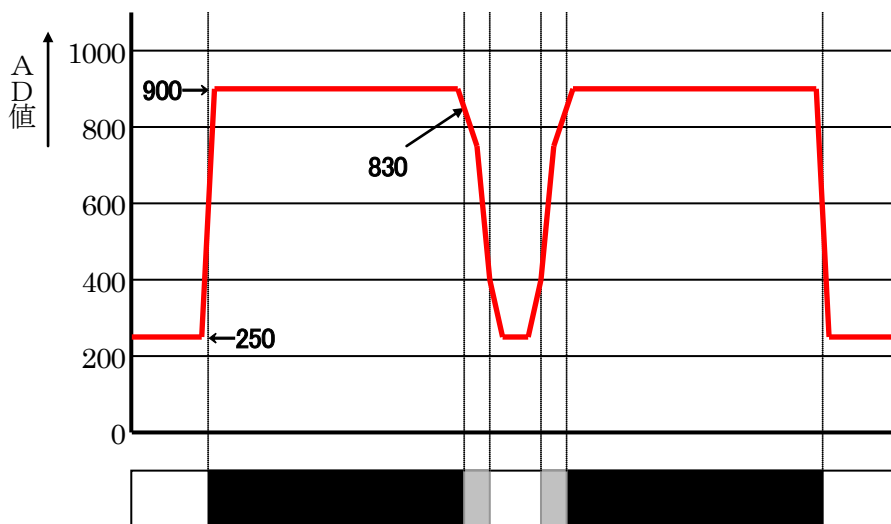
- アナログ入力端子は、ポート 0 の bit0~7、ポート 1 の bit0~3、ポート 7 の bit0~7 の 20 端子ある
※ただし、RY_R8C38 ボードはポート 1 の bit0~3 は、ディップスイッチに繋がっています。
- 0~5V(マイコンボードの電源電圧)の電圧を、0~1023($2^{10}-1$)の値に変換する
- 1 度に A/D 変換できるのは 1 端子のみ、どの端子電圧を A/D 変換するかはプログラムで設定する

A/D 変換について、詳しくは「マイコン実習マニュアル(R8C/38A 版)」のプロジェクト「ad」を参照してください。

2.7 コースの状態を取り込む



上記のような回路を組み、コースとセンサの間隔を約 3mm 一定にしてコースの端から端までずらしていき、そのときの A/D 取得値を簡単なグラフにしてみました。



白色が 250、黒色が 900、中心の黒、灰、白部分は 900 から 250 へ変化していきます。正確には比例していませんが、ほぼ比例していると考えて差し支えありません。

マイコンカーはコース中心の白色と灰色をトレースするので、この部分を詳しく見てみます。コース中心部分は 250、ずれるにしたがって値が大きくなり、黒色部分では約 900 になります。A/D 値をチェックすることにより、中心部分のずれを細かく知ることができます。

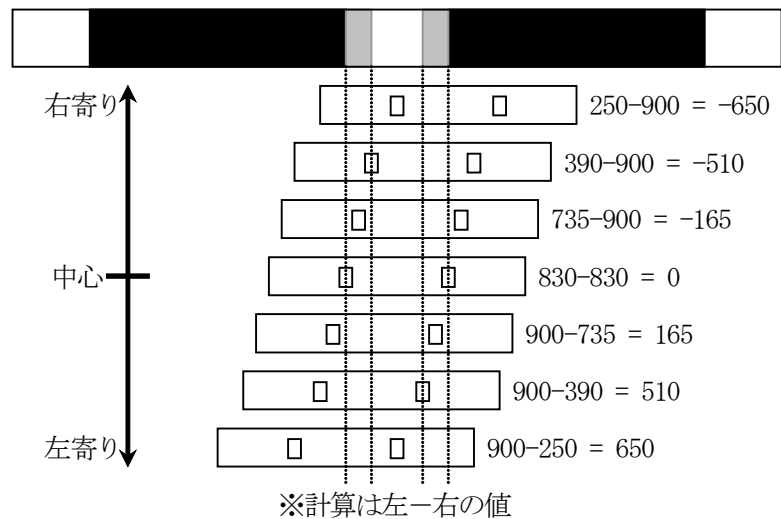
2. アナログセンサ基板 TypeS Ver.2

ただし、1つ問題があります。例えば黒色と灰色のちょうど境目は 830 くらいですが、数字を見ただけでは右側か左側か分かりません。アナログセンサ 1 個では右にずれているのか左にずれているのか分からないのです。

そこで、アナログセンサを 2 個、40mm の間隔で取り付け、次の計算を行います。

センサの値 = 左センサの値 - 右センサの値

この計算を行うことによりセンサの値は、左側にずれているなら正の数、右側なら負の数となり、左右のどちら側に寄っているのか分かります。



今回の例では、センサの値が-650 から 650 まで変化します。センサの状態が正の数なら左に寄っていますのでハンドルを右へ、負の数なら右へ寄っていますのでハンドルを左へ曲げます。また値の大きさで、どのくらいの強さで曲げるかを調整することができます。例えば 50 なら弱く曲げる、500 なら強く曲げる、というように制御します。

3. モータドライブ基板 TypeS Ver.3

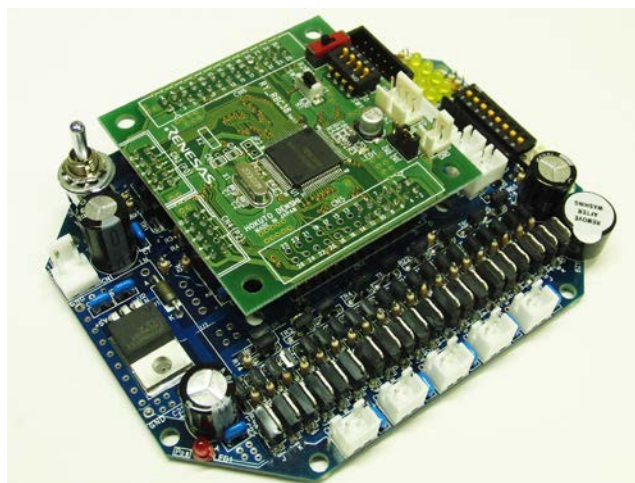
3.1 仕様

モータドライブ基板 TypeS Ver.3 は、モータを 5 個制御することのできる基板です。仕様を下表に示します。

	モータドライブ基板 TypeS Ver. 3	モータドライブ基板 Ver.5 (参考)
対象	既にものづくりを経験されている方が対象	すべての方が対象
部品数	リード線のある部品:約 182 個 表面実装部品:0 個 部品のピンの間隔は 2.54mm 以上	リード線のある部品:約 66 個 表面実装部品:0 個 部品のピンの間隔は 2.54mm 以上
マイコンボードとの接続方法	本基板の上に重ねる	10 芯フラットケーブルにより接続
制御できるモータ数	5 個 自作サーボモータ、左前モータ、 右前モータ、左後モータ、右後モータ	2 個 左モータ、右モータ
モータの制御周期 (PWM 周期)	1[ms]	16[ms] ※ラジコンサーボと共通
制御できるラジコンサーボ	なし	1 個
入力電圧	7.2V 以上 (単三電池 6 本～8 本)	5V±10%、または 7V 以上 (単三電池 4 本～8 本) ただし 7V 以上の電圧を加える場合、LM350 追加セットの追加が必要です
プッシュスイッチ	1 個	1 個
ディップスイッチ	8bit	なし
プログラムで点灯、消灯できる LED	8 個	2 個
リミットスイッチなどの接点入力回路	4 個分(CN6)	なし
エンコーダ入力回路	あり(CN9)	なし
ボリューム入力回路	あり(CN7)	なし
ブザー	あり (周波数は固定です、圧電ブザーではありません)	なし
センサ基板の信号入力コネクタ	あり アナログセンサ基板 TypeS Ver.2、 またはセンサ基板 Ver.4.1 または Ver.5	なし ※RY_R8C38 ボードの CN3(ポート 0)にセンサ基板 Ver.5 を接続します
基板外形	110×90×厚さ 1.6mm	80×75×厚さ 1.6mm
重量 (基板のみ)	約 30g	約 15g
完成時の寸法 (実寸)	幅 110×奥行き 90×高さ 35mm ※スイッチを除くと高さ 22mm	幅 80×奥行き 65×高さ 20mm
重量 (完成品の実測)	約 81g (RY_R8C38 ボードは除く) ※リード線の長さや半田の量で変わります ※参考:RY_R8C38 ボード込みで実測 102g	約 35g ※リード線の長さや半田の量で変わります

詳しくは、モータドライブ基板 TypeS Ver.3 製作マニュアルを参照してください。

3. モータドライブ基板 TypeS Ver.3



▲モータドライブ基板 TypeS Ver.3

3.2 ラジコンサーボと自作サーボ

マイコンカーで使用する場合の、ラジコンサーボと自作サーボの特徴を下記に示します。

項目	ラジコンサーボ	自作サーボ
制御周期	サーボに加える PWM 周期が制御周期 標準的なサーボで 16ms、 デジタルサーボで 5ms 秒速 4m/s なら 16ms で 64mm 進んでしまう	プログラムで可変可能 サンプルプログラムは 1ms 秒速 4m/s なら 1ms で 4mm しか進まない！
モータドライブ回路	サーボに内蔵のため不要	必要
プログラム	PWM のデューティ比を変えるだけ、簡単	現在の角度と目標の角度から、加える PWM のデューティ比を計算、調整が難しい
現在の角度検出	できない ただし別途ボリューム、またはロータリエンコーダを付けることにより可能	ボリューム、またはロータリエンコーダ必要
モータ	サーボに内蔵のため不要 逆に言えば選べない	自分で選定する
ギヤ比	サーボ固有のギヤ比	自分で組む必要があるが、 組み方により自由に設定できる

アナログセンサと自作サーボは基本的にペアで使用します。それは、「**制御周期**」に関わりがあります。

せっかくアナログセンサで中心からのずれが細かく分かっていても、サーボを制御する間隔が長ければ意味がありません。そこで自由に制御周期を設定できるように、サーボ機構を自作します。

自作サーボはギヤ比、モータ、制御周期を自分で選定できるため、高価なラジコンサーボ以上の性能を出すことも可能です。

※サーボについて

サーボは、「物体の位置、方位、姿勢など(機械量)を制御量とし、目標値の任意の変化に追従するように構成された制御系。」(出典:Wikipedia)という意味です。

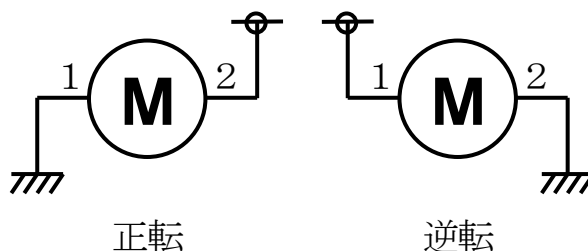
本書では、ラジコン屋さんで販売されている PWM 波形を加えると自動で動くサーボを**ラジコンサーボ**、マイコンで直接モータを制御するサーボを**自作サーボ**と使い分けています。どちらもサーボではありますが、マイコンでの制御方法が大きく異なります。

3.3 モータドライブ回路

3.3.1 モータの回し方(電圧と動作の関係)

マイコンカーを制御するには、モータを「正転、逆転、停止」させる必要があります。これらの状態は、モータの端子 1、端子 2 に加える電圧を変えることにより制御します。

動作	端子 1	端子 2
正転	GND 接続	+接続
逆転	+接続	GND 接続
停止	後述	

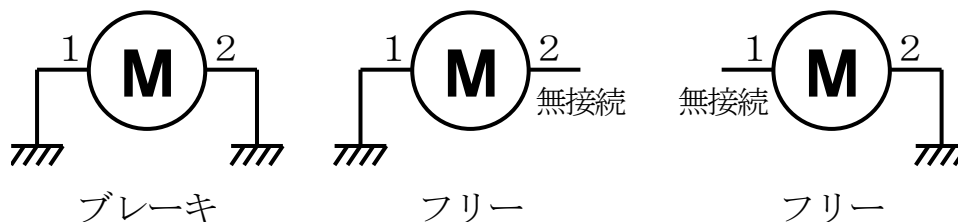


停止には、ブレーキとフリーの2種類あります。

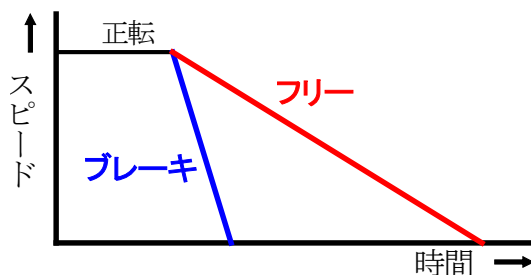
ブレーキは、端子間をショートさせモータの発電作用(逆起電圧)を利用しモータを素早く止める方法です。

フリーは、モータの端子 1、または端子 2 のどちらか(または両方)を無接続にすることにより、惰性でモータの回転が遅くなる動作をいいます。

動作	端子 1	端子 2
ブレーキ	GND 接続	GND 接続
フリー	+接続または GND 接続	無接続
フリー	無接続	+接続または GND 接続



正転状態からブレーキ動作にしたとき、フリー動作にしたときのスピードの落ち方の違いを下図に示します。



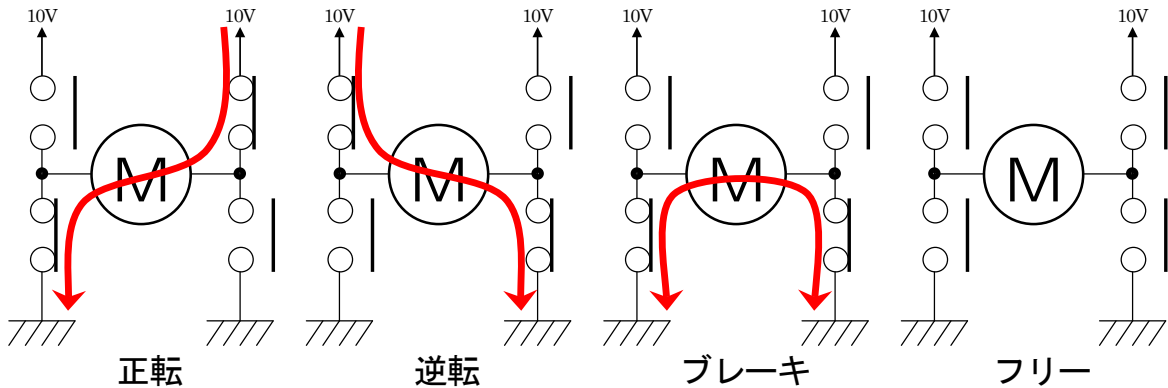
フリーはブレーキと比べ、スピードの減速が緩やかです。フリーは、スピードをゆっくり落としたい場合などに使用します。

3. モータドライブ基板 TypeS Ver.3

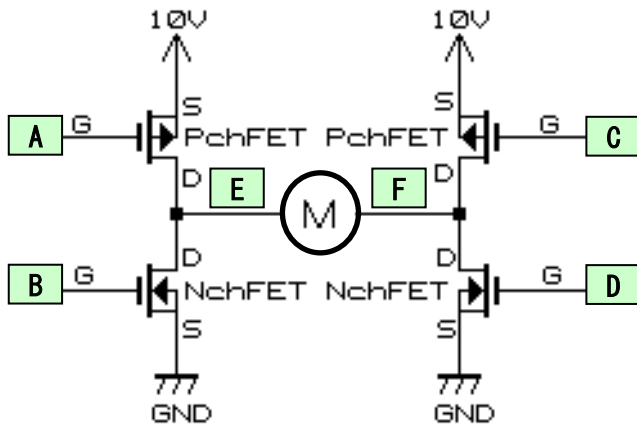
3.3.2 Hブリッジ回路

モータを正転、逆転、ブレーキ、フリーにするには下図のように、モータを中心としてH型に4つのスイッチを付けます。その形から「Hブリッジ回路」と呼ばれています。

この4つのスイッチをそれぞれ ON/OFF することにより、正転、逆転、ブレーキ、フリー制御を行います。



3.3.3 スイッチをFETにする



実際の回路では、前記のスイッチを FET で行います。電源のプラス側に P チャネル FET (2SJ タイプ)、マイナス側に N チャネル FET (2SK タイプ) を使用します。

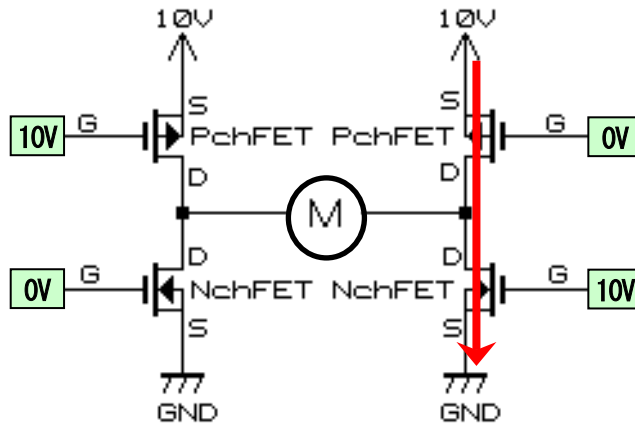
P チャネル FET は、 V_G (ゲート電圧) $<$ V_S (ソース電圧) のとき、D-S (ドレイン-ソース) 間に電流が流れます。

N チャネル FET は、 V_G (ゲート電圧) $>$ V_S (ソース電圧) のとき、D-S (ドレイン-ソース) 間に電流が流れます。

これら4つのFETのゲートに加える電圧を変えることにより、正転、逆転、ブレーキ、フリーの動作を行います。下表にFET A~Dのゲートに0Vまたは10Vを加えたときの動作を示します。

A	B	C	D	FET A の動作	FET B の動作	FET C の動作	FET D の動作	E の電圧	F の電圧	モータ動作
0V	0V	0V	0V	ON	OFF	ON	OFF	10V	10V	ブレーキ
0V	0V	0V	10V	ON	OFF	ON	ON	10V	ショート!	設定不可
0V	0V	10V	0V	ON	OFF	OFF	OFF	10V	フリー	フリー
0V	0V	10V	10V	ON	OFF	OFF	ON	10V	0V	逆転
0V	10V	0V	0V	ON	ON	ON	OFF	ショート!	10V	設定不可
0V	10V	0V	10V	ON	ON	ON	ON	ショート!	ショート!	設定不可
0V	10V	10V	0V	ON	ON	OFF	OFF	ショート!	フリー	設定不可
0V	10V	10V	10V	ON	ON	OFF	ON	ショート!	0V	設定不可
10V	0V	0V	0V	OFF	OFF	ON	OFF	フリー	10V	フリー
10V	0V	0V	10V	OFF	OFF	ON	ON	フリー	ショート!	設定不可
10V	0V	10V	0V	OFF	OFF	OFF	OFF	フリー	フリー	フリー
10V	0V	10V	10V	OFF	OFF	OFF	ON	フリー	0V	フリー
10V	10V	0V	0V	OFF	ON	ON	OFF	0V	10V	正転
10V	10V	0V	10V	OFF	ON	ON	ON	0V	ショート!	設定不可
10V	10V	10V	0V	OFF	ON	OFF	OFF	0V	フリー	フリー
10V	10V	10V	10V	OFF	ON	OFF	ON	0V	0V	ブレーキ

「設定不可」の部分は、ショート状態となるため、設定してはいけません。例えば、A=10V、B=0V、C=0V、D=10V のとき、下図のように左側の P チャネル FET と N チャネル FET が 10V から 0V まで直接つながり、ショートと同じ状態になってしまいます。



3.3.4 スピード制御

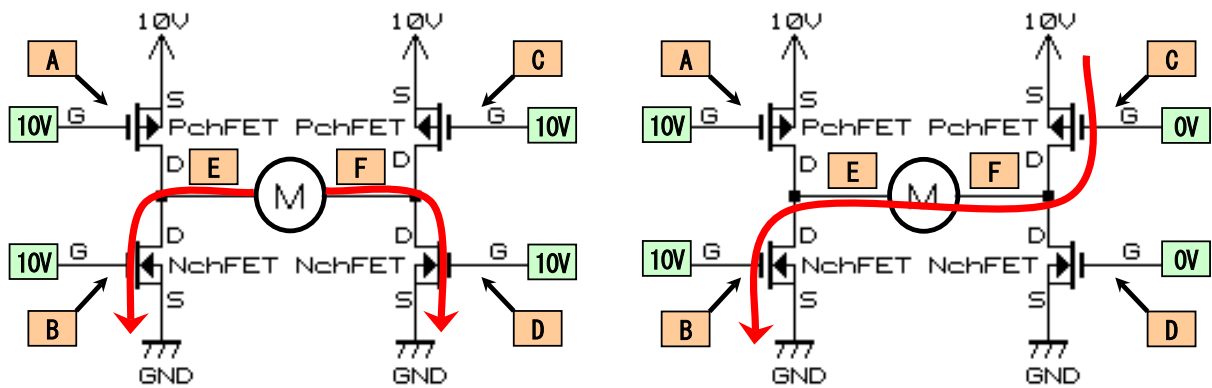
正転するスピードを変えたい場合、「正転→ブレーキ→正転→ブレーキ…」を高速に繰り返します。今回のサンプルプログラムは、「正転→ブレーキ」を 1ms 間で行い、正転とブレーキの割合を変えることでスピードを変えます。

正転とブレーキを繰り返すときの 4 つの FET のゲート電圧は、下表のようになります。

A	B	C	D	FET A の動作	FET B の動作	FET C の動作	FET D の動作	E の電圧	F の電圧	モータ動作
10V	10V	10V	10V	OFF	ON	OFF	ON	0V	0V	ブレーキ
10V	10V	0V	0V	OFF	ON	ON	OFF	0V	10V	正転

●ブレーキ動作

●正転動作



表より FET C と FET D のゲート電圧を、0V と 10V に変えればよいことが分かります。

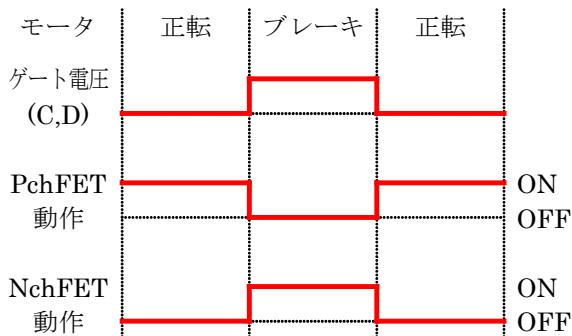
3. モータドライブ基板 TypeS Ver.3

3.3.5 正転とブレーキの切り替え時にショートしてしまう

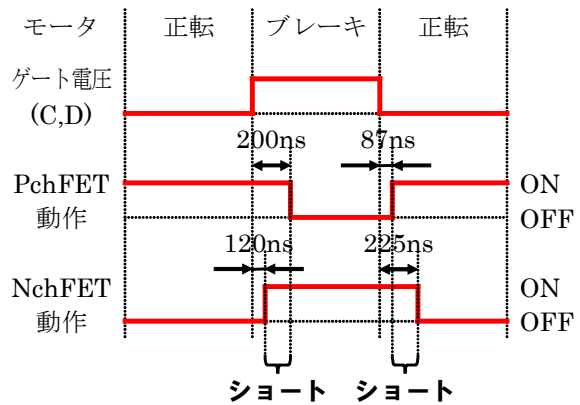
前記の回路を実際に組んで動作させると、FET が非常に熱くなりました。どうしてでしょうか。

FET のゲートから信号を入力し、ドレイン・ソース間が ON/OFF するとき、左下図「理想的な波形」のように、P チャンネル FET と N チャンネル FET が反応してブレーキと正転がスムーズに切り替わるように思えます。しかし、実際にはすぐには動作せず遅延時間があります。この遅延時間は FET が OFF→ON のときより、ON→OFF のときの方が長くなっています。そのため、右下図「実際の波形」のように、短い時間ですが両 FET が ON 状態となり、ショートと同じ状態になってしまいます。

理想的な波形



実際の波形



ON してから実際に反応し始めるまでの遅延を「ターン・オン遅延時間」、ON になり初めてから実際に ON するまでを「上昇時間」、OFF してから実際に反応し始めるまでの遅延を「ターン・オフ遅延時間」、OFF になり初めてから実際に OFF するまでを「下降時間」といいます。

実際に OFF→ON するまでの時間は「ターン・オン遅延時間 + 上昇時間」、ON→OFF するまでの時間は「ターン・オフ遅延時間 + 下降時間」となります。「実際の波形」の図に出ている遅れの時間は、これらの時間のことで

参考までにルネサス エレクトロニクス(株)製の FET 2SJ530 と 2SK2869 の電気的特性を下記に示します。

2SJ530(P チャンネル)

電気的特性						
(Ta=25°C)						
項目	記号	Min	Typ	Max	単位	測定条件
ドレイン・ソース破壊電圧	$V_{(BR)DSS}$	-60	—	—	V	$I_D = 10mA, V_{GS} = 0$
ゲート・ソース破壊電圧	$V_{(BR)GSS}$	±20	—	—	V	$I_D = ±100μA, V_{DS} = 0$
ドレイン遮断電流	I_{DSS}	—	—	-10	μA	$V_{DS} = -60V, V_{GS} = 0$
ゲート遮断電流	I_{GSS}	—	—	±10	μA	$V_{GS} = ±16V, V_{DS} = 0$
ゲート・ソース遮断電圧	$V_{GS(off)}$	-1.0	—	-2.0	V	$V_{DS} = 10V, I_D = 1mA$
順伝達アドミタンス	$ y_{fs} $	6.5	11	—	S	$I_D = 8A, V_{DS} = 10V^{①}$
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.08	0.10	Ω	$I_D = 8A, V_{GS} = 10V^{①}$
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.11	0.16	Ω	$I_D = 8A, V_{GS} = 4V^{①}$
入力容量	C_{iss}	—	850	—	pF	$V_{DS} = -10V, V_{GS} = 0$
出力容量	C_{oss}	—	420	—	pF	$f = 1MHz$
掃蕩容量	C_{rss}	—	110	—	pF	
ターン・オン遅延時間	$t_d(on)$	—	12	—	ns	$V_{GS} = -10V, I_D = 8A$
上昇時間	t_r	—	75	—	ns	$R_L = 3.75Ω$
ターン・オフ遅延時間	$t_d(off)$	—	125	—	ns	
下降時間	t_f	—	75	—	ns	
ダイオード順電圧	V_{SD}	—	-1.1	—	V	$I_S = 15A, V_{GS} = 0$
逆回復時間	t_{rr}	—	70	—	ns	$I_S = 15A, V_{GS} = 0$ $dI/dt = 50A/μs$

注) 4. パルス測定

OFF→ON は
87ns 遅れる

ON→OFF は
200ns 遅れる

2SK2869 (Nチャンネル)

電気的特性						
(Ta=25°C)						
項目	記号	Min	Typ	Max	単位	測定条件
ドレイン・ソース破壊電圧	$V_{(BR)DSS}$	60	—	—	V	$I_D = 10mA, V_{GS} = 0$
ゲート・ソース破壊電圧	$V_{(BR)GSS}$	±20	—	—	V	$I_D = ±100μA, V_{DS} = 0$
ドレイン遮断電流	I_{DSS}	—	—	10	μA	$V_{DS} = 60V, V_{GS} = 0$
ゲート遮断電流	I_{GSS}	—	—	±10	μA	$V_{GS} = ±16V, V_{DS} = 0$
ゲート・ソース遮断電圧	$V_{GS(off)}$	1.5	—	2.5	V	$V_{DS} = 10V, I_D = 1mA$
順伝達アドミタンス	$ y_{fs} $	10	16	—	S	$I_D = 10A, V_{DS} = 10V^{①}$
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.033	0.045	Ω	$I_D = 10A, V_{GS} = 10V^{①}$
ドレイン・ソースオン抵抗	$R_{DS(on)}$	—	0.055	0.07	Ω	$I_D = 10A, V_{GS} = 4V^{①}$
入力容量	C_{iss}	—	740	—	pF	$V_{DS} = 10V, V_{GS} = 0$
出力容量	C_{oss}	—	380	—	pF	$f = 1MHz$
掃蕩容量	C_{rss}	—	140	—	pF	
ターン・オン遅延時間	$t_d(on)$	—	10	—	ns	$V_{GS} = 10V, I_D = 10A$
上昇時間	t_r	—	110	—	ns	$R_L = 3Ω$
ターン・オフ遅延時間	$t_d(off)$	—	135	—	ns	
下降時間	t_f	—	120	—	ns	
ダイオード順電圧	V_{SD}	—	-1.0	—	V	$I_S = 20A, V_{GS} = 0$
逆回復時間	t_{rr}	—	40	—	ns	$I_S = 20A, V_{GS} = 0$ $dI/dt = 50A/μs$

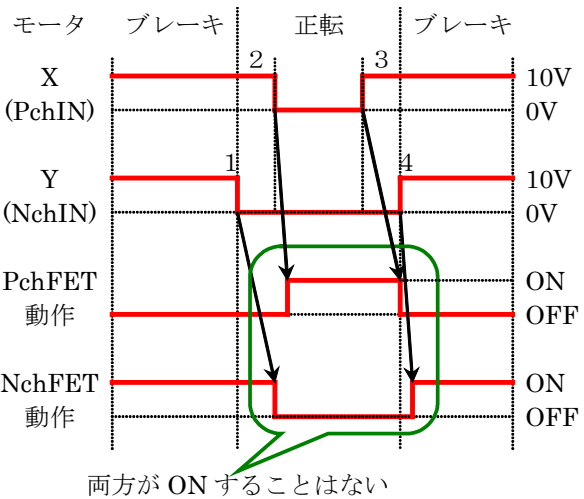
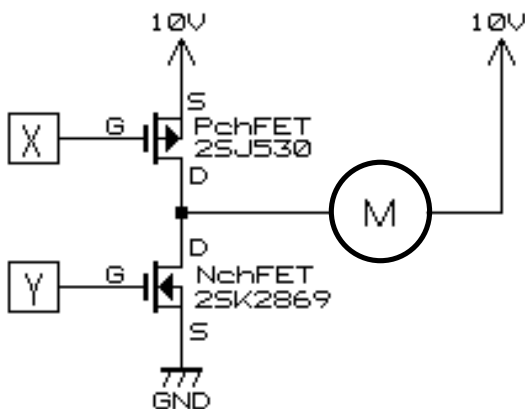
注) 1. パルス測定

OFF→ON は
120ns 遅れる

ON→OFF は
225ns 遅れる

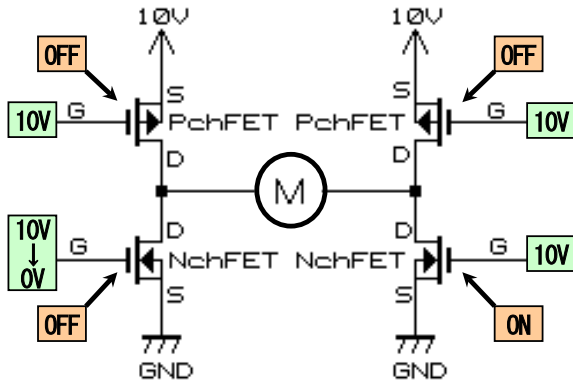
3.3.6 PチャンネルとNチャンネルの短絡防止回路

解決策としては、先ほどの回路図にあるPチャンネルFETとNチャンネルFETを同時にON/OFF するのではなく、少し時間をずらしてON/OFF させ、ショートさせないようにします。



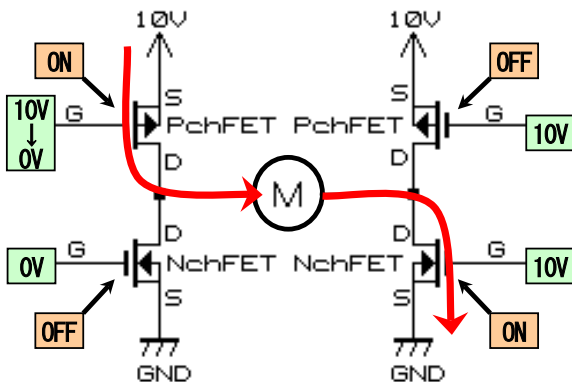
3. モータドライブ基板 TypeS Ver.3

(1) チャネル FET を OFF にする(フリー状態)



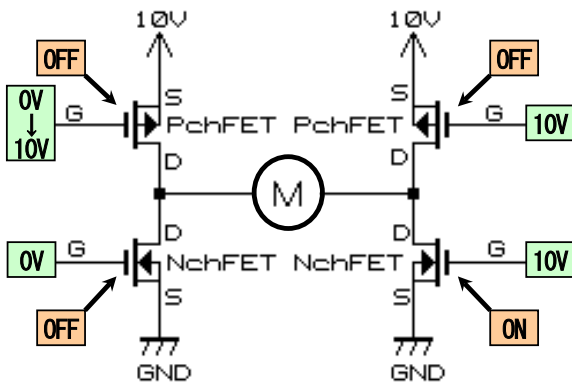
P チャネル FET を ON する前より先に、N チャネル FET のゲート電圧を 10V→0V にします。225ns 後に FET は OFF になります。フリー状態です。

(2) P チャネル FET を ON にする(正転状態)



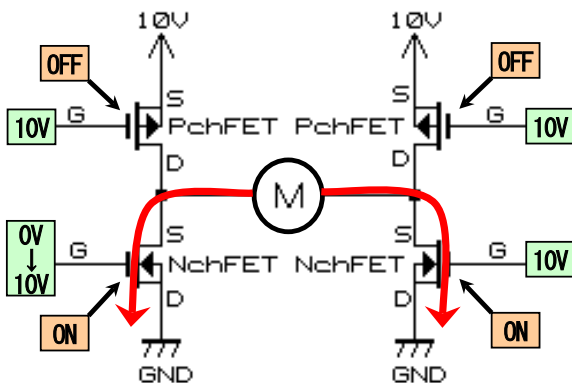
次に、P チャネル FET のゲート電圧を 10V→0V にします。87ns 後に FET は ON します。正転状態です。

(3) P チャネル FET を OFF にする(フリー状態)



次に、P チャネル FET のゲート電圧を 0V→10V にします。200ns 後に FET は OFF します。フリー状態です。

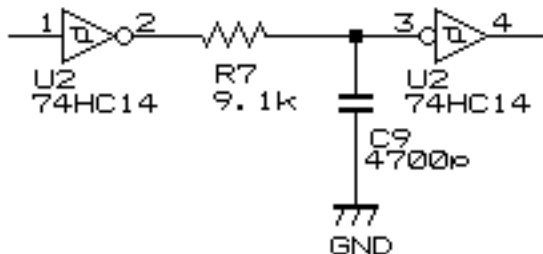
(4) N チャネル FET を ON にする(ブレーキ状態)



次に、N チャネル FET のゲート電圧を 0V→10V にします。120ns 後に FET は ON します。ブレーキ状態です。

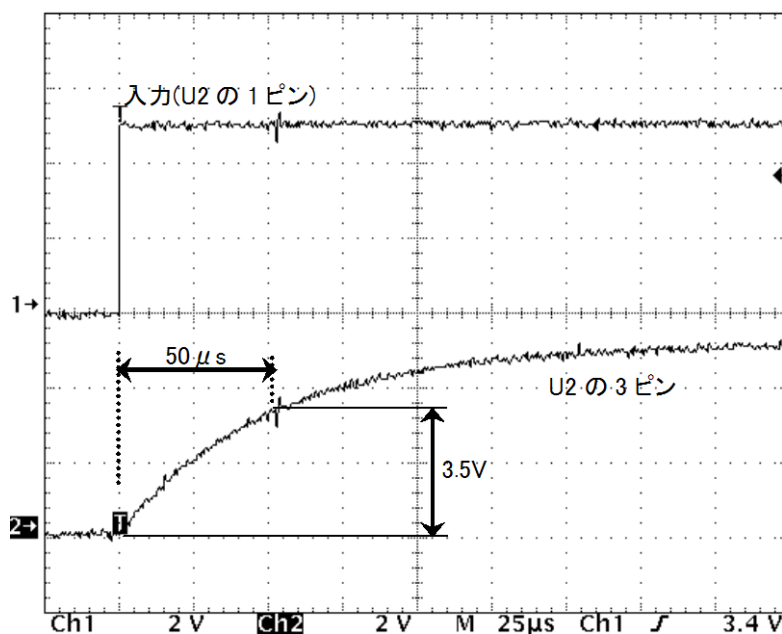
3.3.7 PチャネルとNチャネルの短絡防止回路

モータドライブ基板 TypeS Ver.3 は、コンデンサ、抵抗を使った遅延回路で短絡を防止しています。遅延時間は、下図のような積分回路で作ります。積分回路については、多数の専門書があるので、そちらを参照して下さい。



遅延時間はほぼ
 時定数 $T = CR$ [s]
 で計算することができます。
 今回は $9.1k\Omega$ 、 $4700pF$ なので、計算すると
 $T = (9.1 \times 10^3) \times (4700 \times 10^{-12})$
 $= 42.77 [\mu s]$
 となります。

入力(U2の1ピン)と出力(U2の3ピン)の関係を下図に示します。



74HC シリーズは 3.5V 以上の入力電圧があると“1”とみなします。

実際に波形を観測し、3.5V になるまでの時間を計ると約 $50 \mu s$ になりました。

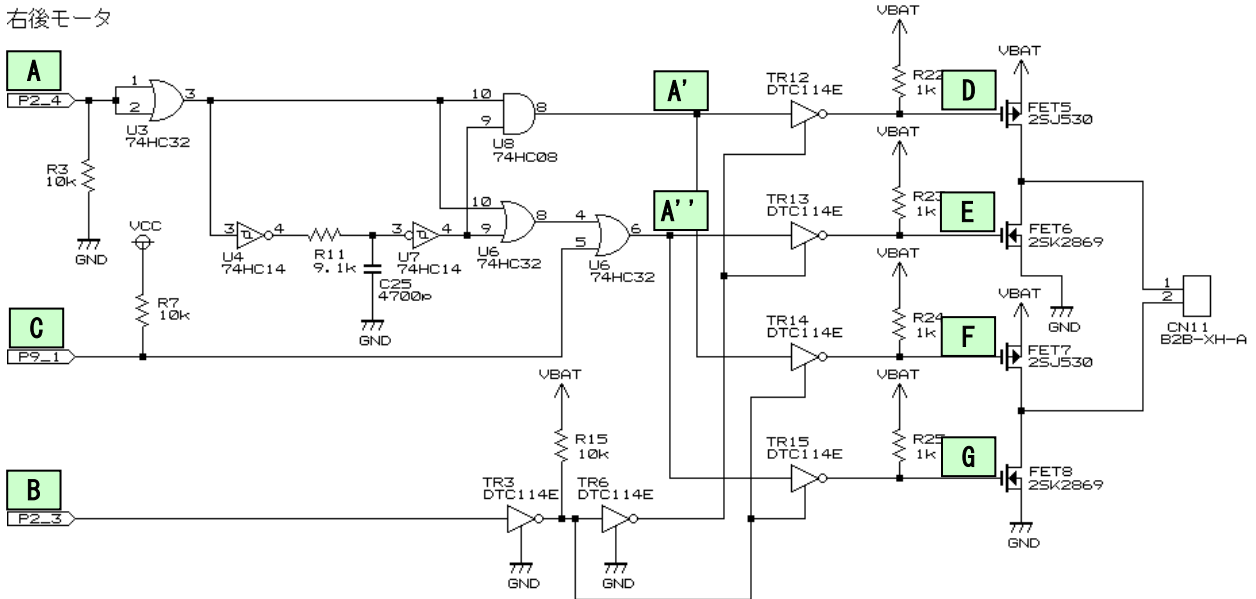
先ほどの「実際の波形」の図では最高でも 225ns のずれしかありませんが、今回の積分回路では $50 \mu s$ の遅延時間を作っています。これは、FET 以外にも、電圧変換用の抵抗内蔵トランジスタの遅延時間、FET のゲートのコンデンサ成分による遅れなどを含めたためです。

3. モータドライブ基板 TypeS Ver.3

3.3.8 実際の回路

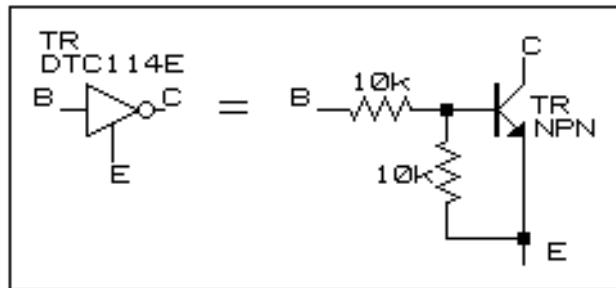
右後モータ回路を例に説明します。P2_4(A)がPWM出力端子、P2_3(B)が正転／逆転切り替え端子、P9_1(C)がブレーキ／フリー切り替え端子です。

右後モータ

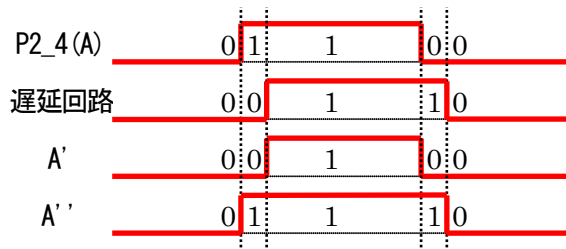


※ VBAT = 10V とする

DTC114E は抵抗内蔵トランジスタと呼び、下記のようにトランジスタ 1 個と抵抗 2 個が内蔵された回路です。



A'は信号 A と遅延回路の AND、A''は信号 A と遅延回路の OR の信号です。A、A'、A''の波形は下記ようになります。

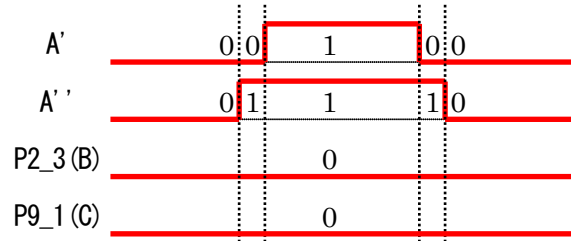


※0 : 0V、1 : 5V

3.3.9 正転、ブレーキ時の動作

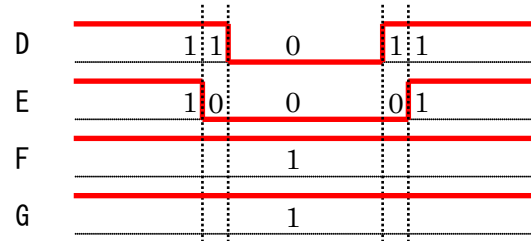
正転(P2_3="0"), ブレーキ(P9_1="0") のとき、P2_4 端子に 1 パルス入ったときの波形と各端子の状態は下記のようになります。

ポート出力



※0 : 0V、1 : 5V

FET のゲート電圧

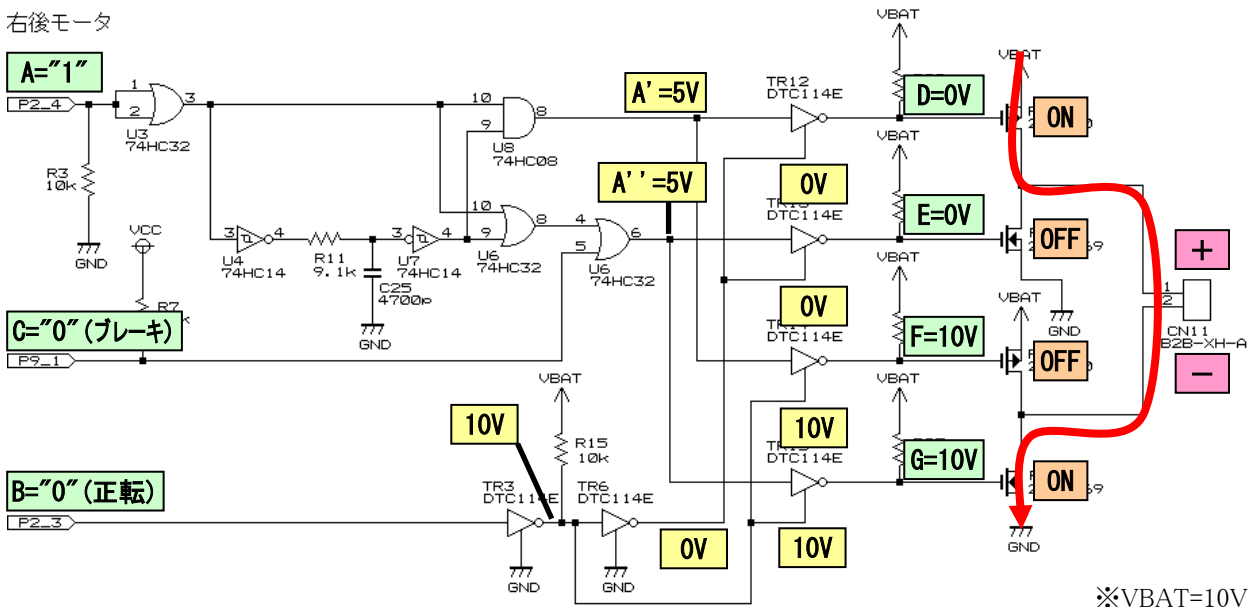


※0 : 0V、1 : 10V

A'	A''	B	C	D	E	F	G	モータ動作
0 (0V)	0 (0V)			10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	ブレーキ 両端子 GND
0 (0V)	1 (5V)			10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー 無接続状態
1 (5V)	1 (5V)	0 (0V)	0 (0V)	0V (ON)	0V (OFF)	10V (OFF)	10V (ON)	正転 CN11.1=10V CN11.2=0V
0 (0V)	1 (5V)			10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー 無接続状態
0 (0V)	0 (0V)			10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	ブレーキ 両端子 GND

A="1", B="0"(正転)、C="0"(ブレーキ)のときの動作を下記に示します。

右後モータ

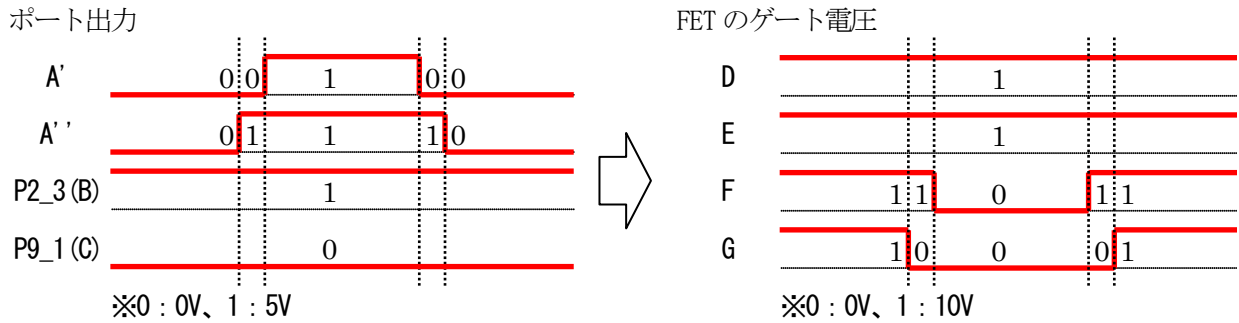


※VBAT=10V

3. モータドライブ基板 TypeS Ver.3

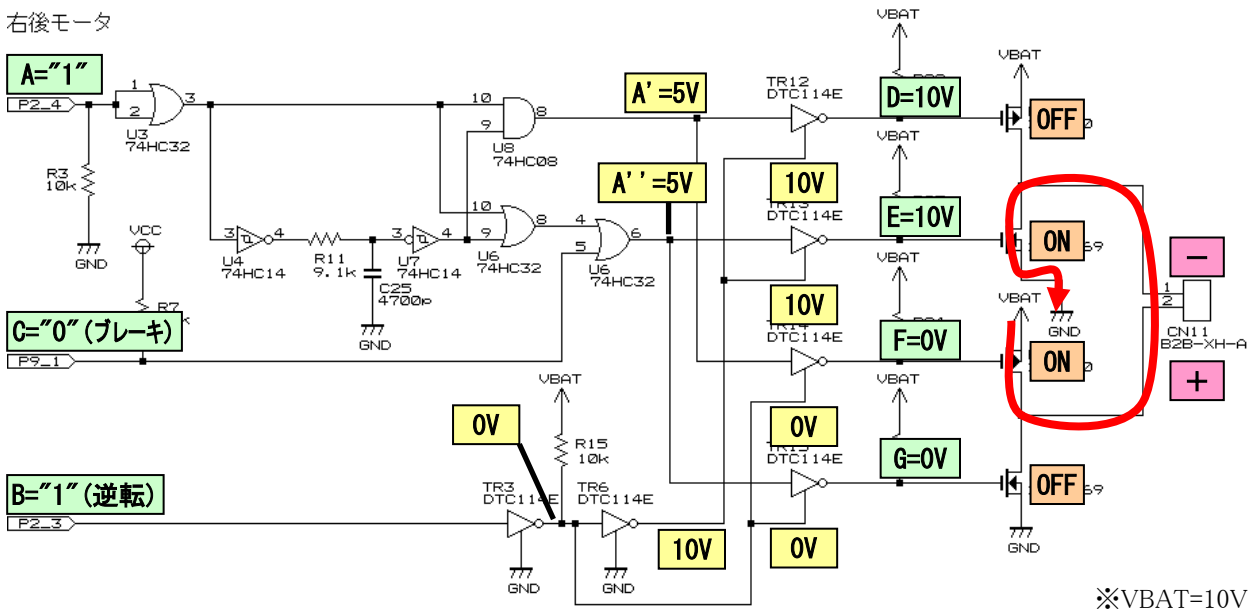
3.3.10 逆転、ブレーキ時の動作

逆転(P2_3="1")、ブレーキ(P9_1="0")のとき、P2_4 端子に1パルス入ったときの波形と各端子の状態は下記のようになります。



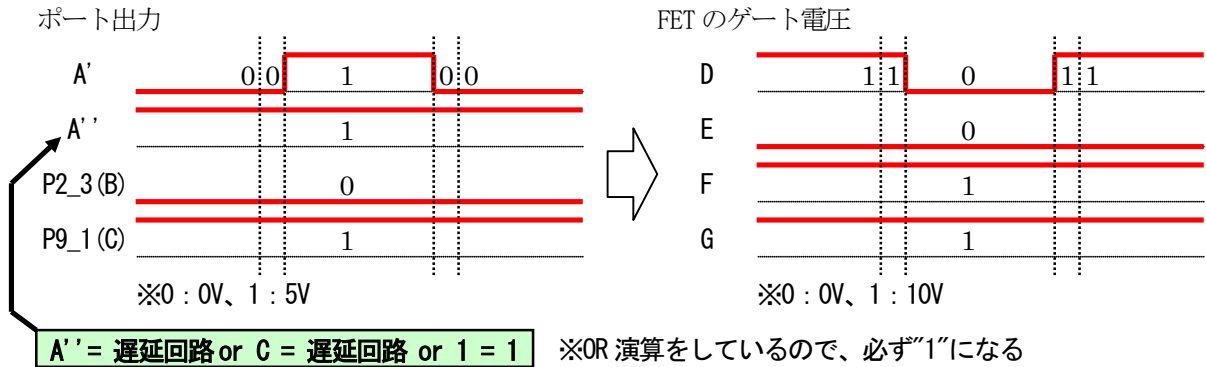
A'	A''	B	C	D	E	F	G	モータ動作
0 (0V)	0 (0V)	1 (5V)	0 (0V)	10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	ブレーキ 両端子 GND
0 (0V)	1 (5V)			10V (OFF)	10V (ON)	10V (OFF)	0V (OFF)	フリー 無接続状態
1 (5V)	1 (5V)			10V (OFF)	10V (ON)	0V (ON)	0V (OFF)	逆転 CN11.1=0V CN11.2=10V
0 (0V)	1 (5V)			10V (OFF)	10V (ON)	10V (OFF)	0V (OFF)	フリー 無接続状態
0 (0V)	0 (0V)			10V (OFF)	10V (ON)	10V (OFF)	10V (ON)	ブレーキ 両端子 GND

A="1"、B="1"(逆転)、C="0"(ブレーキ)のときの動作を下記に示します。



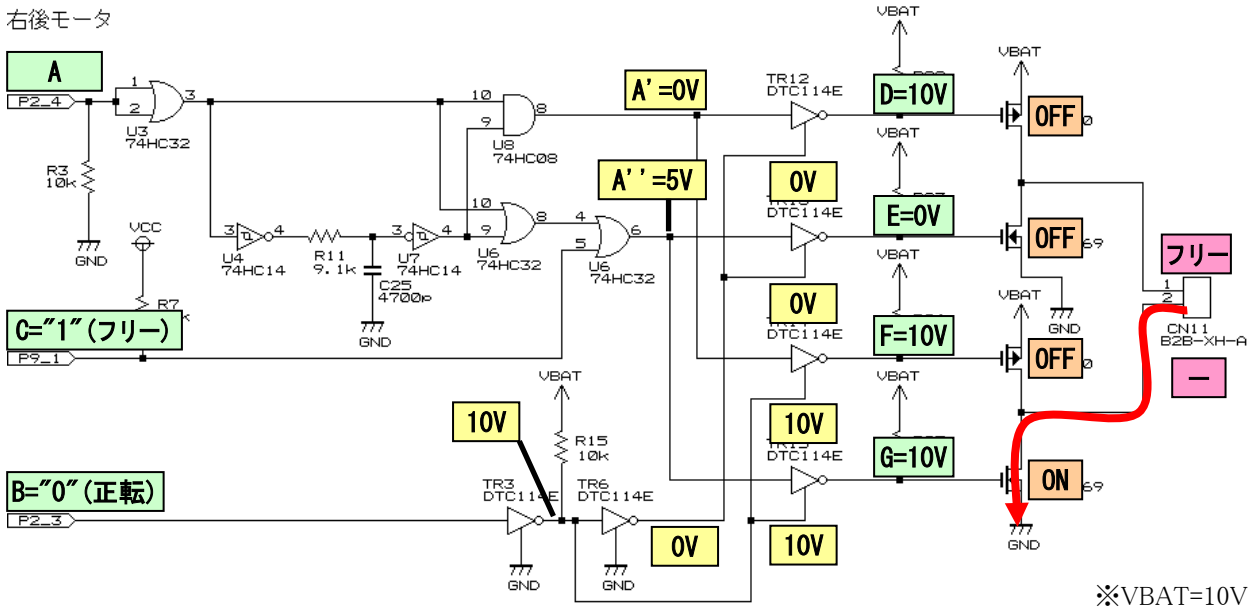
3.3.11 正転、フリー時の動作

正転(P2_3="0"), フリー(P9_1="1") のとき、P2_4 端子に 1 パルス入ったときの波形と各端子の状態は下記のようになります。



A'	A''	B	C	D	E	F	G	モータ動作
0 (0V)	1 (5V)	0 (0V)	1 (5V)	10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー 無接続状態
1 (5V)	1 (5V)			0V (ON)	0V (OFF)	10V (OFF)	10V (ON)	正転 CN11.1=10V CN11.2=0V
0 (0V)	1 (5V)			10V (OFF)	0V (OFF)	10V (OFF)	10V (ON)	フリー 無接続状態

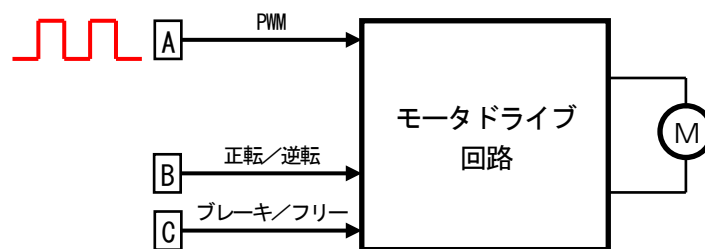
A'="0"、A'="1"、B="0"(正転)、C="1"(フリー)のときの動作を下記に示します。



3. モータドライブ基板 TypeS Ver.3

3.3.12 マイコンのポート割り振り

5 個の各モータは、下図のように 3 本で制御しています。



各モータを制御するマイコンのポートを下表に示します。

モータ	コネクタ	PWM 端子	正転/逆転 切り替え端子	ブレーキ/フリー 切り替え端子
		A	B	C
左後	CN10	P2_2	P2_1	P9_0
右後	CN11	P2_4	P2_3	P9_1
サーボ	CN12	P2_5	P2_6	ブレーキのみ
左前	CN13	P0_5	P2_0	P9_2
右前	CN14	P0_6	P2_7	P9_3

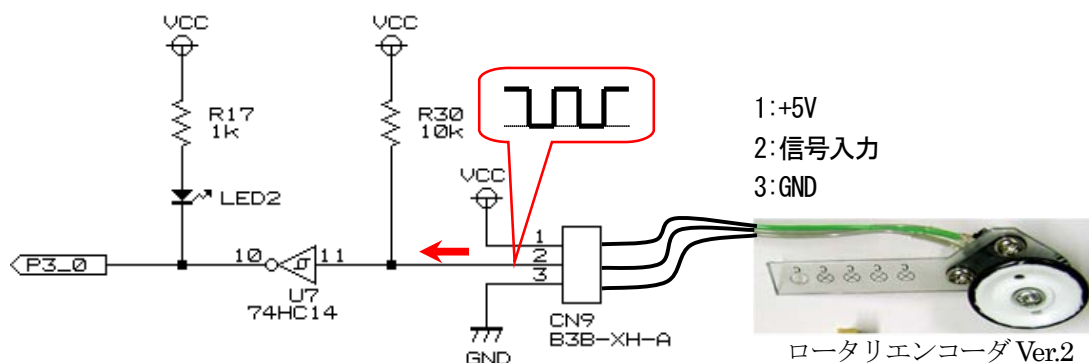
各モータを制御するために 3bit 分の端子を使っています(自作サーボモータは 2bit です)。すべて出力端子にします。

3.4 ロータリエンコーダ信号入力回路

モータドライブ基板 TypeS Ver.3 には、ロータリエンコーダの信号を入力するコネクタがあります。ロータリエンコーダは別売りです。本基板のロータリエンコーダ回路、プログラムの特徴を、下記に示します。

- ロータリエンコーダを接続する専用コネクタ CN9 が実装済み
- 入力信号 10kΩ でプルアップ済み、オープンコレクタ信号でも外付け抵抗の取り付け必要なし
- ロータリエンコーダから入力された信号をシュミット・トリガ NOT 回路(74HC14)にて波形整形
- ロータリエンコーダの信号を LED2 にて確認可能
- ロータリエンコーダのパルス入力端子はマイコンの P3_0 端子から入力

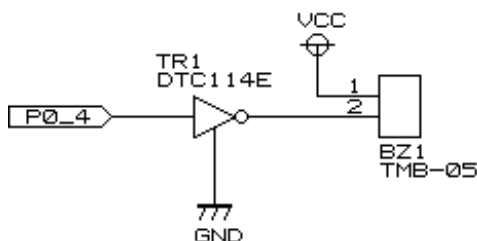
詳しくは、「ロータリエンコーダ kit07_38a プログラム解説マニュアル(R8C/38A 版)」を参照してください。



3.5 ブザー回路

モータドライブ基板 TypeS Ver.3 には、TMB-05 というブザーが実装されています。本基板の回路、サンプルプログラムの特徴を、下記に示します。

- P0_4 端子を"1"にすると抵抗内蔵トランジスタを介して、ブザーに 5V がかかる(ブザーON)
- ブザーに 5V を加えると音が鳴る(回路内蔵型ブザー)
- 音は約 2,300Hz 固定

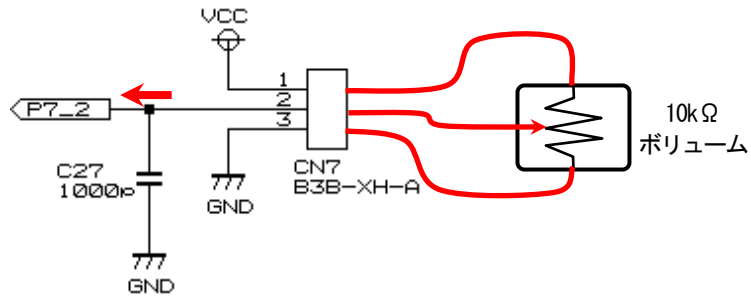


3. モータドライブ基板 TypeS Ver.3

3.6 ボリューム信号入力回路

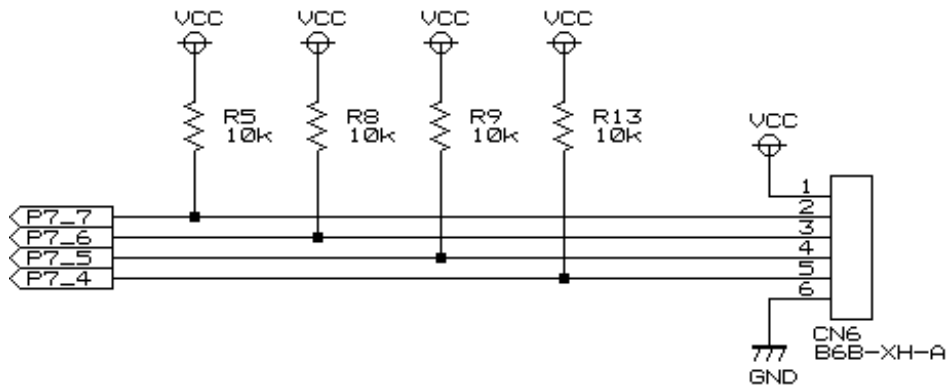
モータドライブ基板 TypeS Ver.3 には、ステアリング角度検出用のボリュームの入力コネクタが実装されています。本基板の回路、プログラムの特徴を下記に示します。

- 3ピン(抵抗の両端と可変部分がある)のボリュームを取り付け可能
- 1000pF のコンデンサをノイズ低減用に実装
- 入力された電圧 0~5V を、R8C/38A マイコンの P7_2 端子で A/D 変換して 0~1023($2^{10}-1$)に変換



3.7 信号入力回路

モータドライブ基板 TypeS Ver.3 には、リミットスイッチなどの 4 個分の信号を入力するコネクタが実装されています。基板側で 10kΩ でプルアップ済みです。



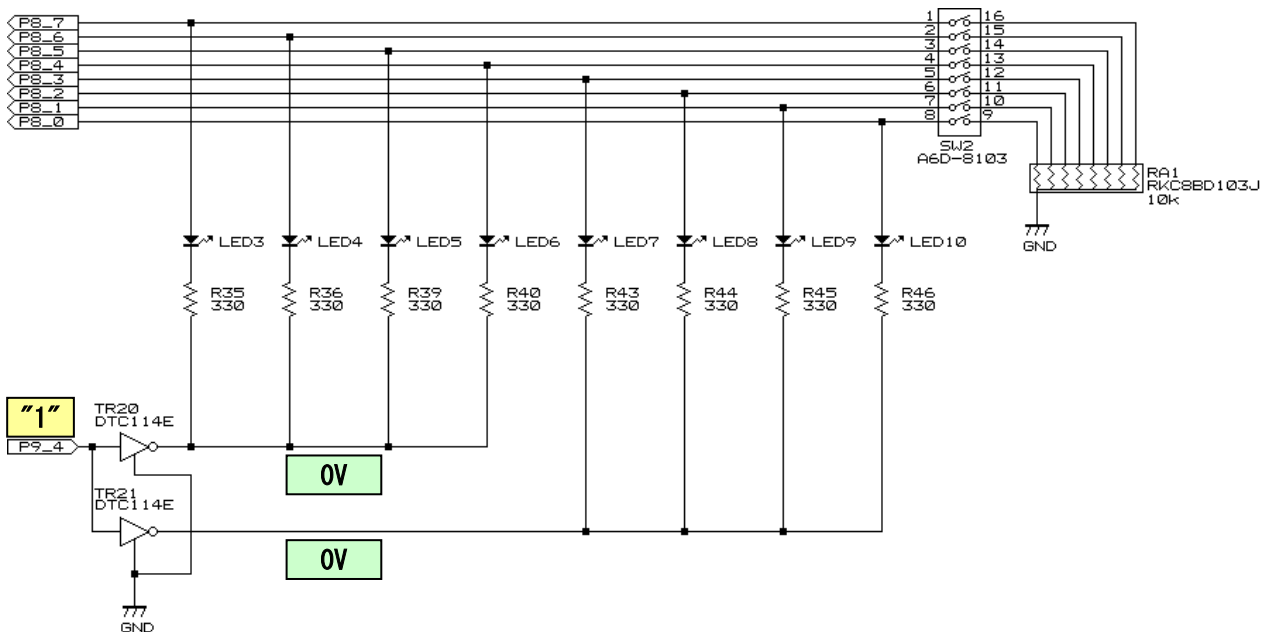
3.8 LED、ディップスイッチ回路

モータドライブ基板 TypeS Ver.3 には、プログラムで点灯／消灯することのできる LED が 8 個、プログラムで状態を読み込める 8bit ディップスイッチが 1 個搭載されています。これらは P8_7～P8_0 端子兼用となっており、プログラムで切り替えて使用します。

3.8.1 LED を点灯させるとき

次の手順で LED を点灯させます。

- ①P9_4 を"1"にして、LED のカソード側を 0V にします。
- ②P8_7～P8_0 端子の入出力設定を出力にします。
- ③LED を点灯させる場合は"1"、消灯させる場合は"0"を P8_7～P8_0 端子から出力します。



LED に流れる電流の計算式を下記に示します。

$$\begin{aligned}
 \text{LED に流れる電流} &= (\text{電源電圧} - \text{LED の直流順電圧}) / \text{電流制限抵抗} \\
 &= (4.7 - 2.2) / 330 \\
 &= 7.6\text{mA}
 \end{aligned}$$

※電源電圧は、抵抗内蔵トランジスタで 0.1～0.3V 程度電圧降下があるため、4.7V としています。

LED3～10 に実装している EMAY3338S(黄)は、20mA まで流せますので、もっと明るくしたい場合は電流制限抵抗の値を小さくしてください。

ディップスイッチは、OFF であれば回路に影響はありません。ON の場合は、10kΩ を通して GND に接続されますが、こちらも LED 点灯回路に影響はありません。

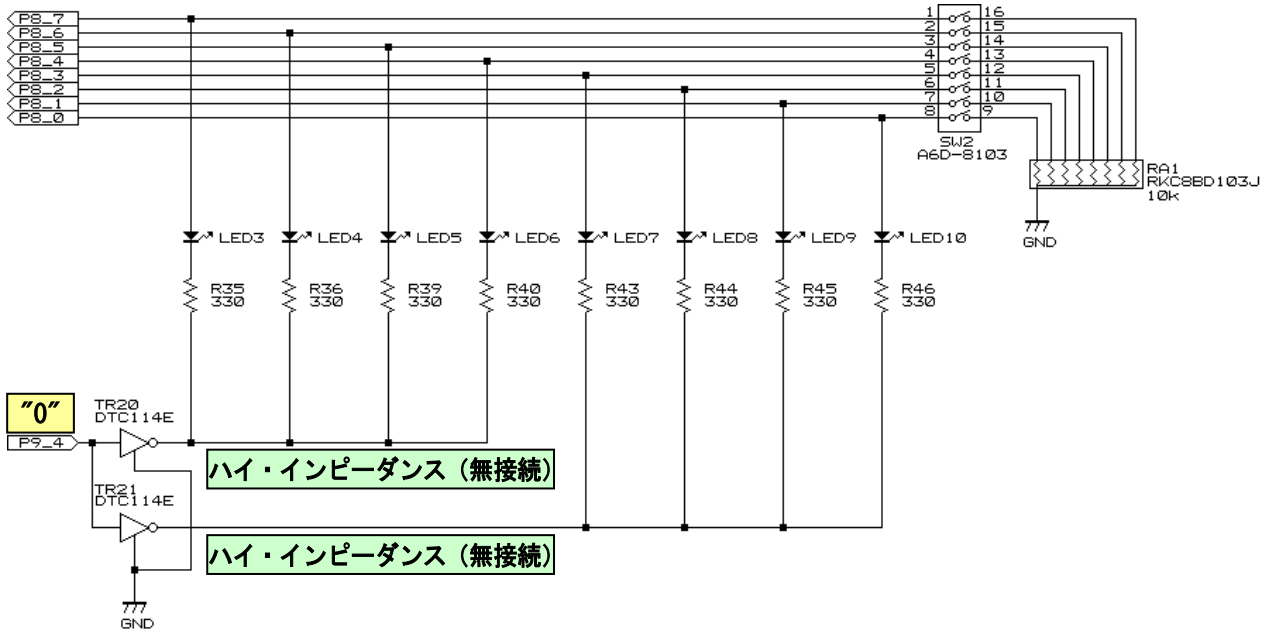
抵抗内蔵トランジスタに流せる電流は 50mA です。LED が 8 個点灯すると、60.8mA (7.6mA×8 個) の電流が流れ定格を超えてしまいます。そのため、LED4 個を抵抗内蔵トランジスタ 1 個で駆動しています。

3. モータドライブ基板 TypeS Ver.3

3.8.2 ディップスイッチの状態を読み込むとき

次の手順でディップスイッチの状態を読み込みます。

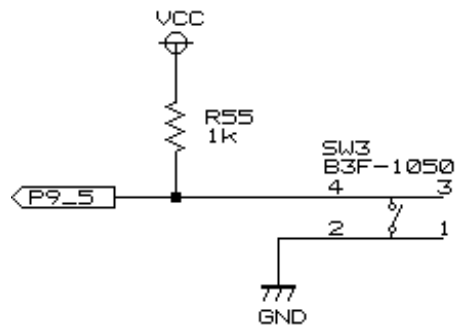
- ①P9_4 を"0"にして、LED のカソード側をハイ・インピーダンス(無接続)状態にします。
 - ②P8_7~P8_0 端子の入出力設定を入力します。
 - ③P8_7~P8_0 端子の R8C/38A マイコン内蔵のプルアップ抵抗 (25~100kΩ、標準で 50kΩ)を ON にします。
 - ④ディップスイッチの状態を読み込みます。
- ※③と④の間は、100 μs 以上あけてください。



ディップスイッチが ON なら 10kΩ を通して"0"が、OFF ならマイコン内蔵のプルアップ抵抗により"1"が入力されます。

3.9 プッシュスイッチ回路

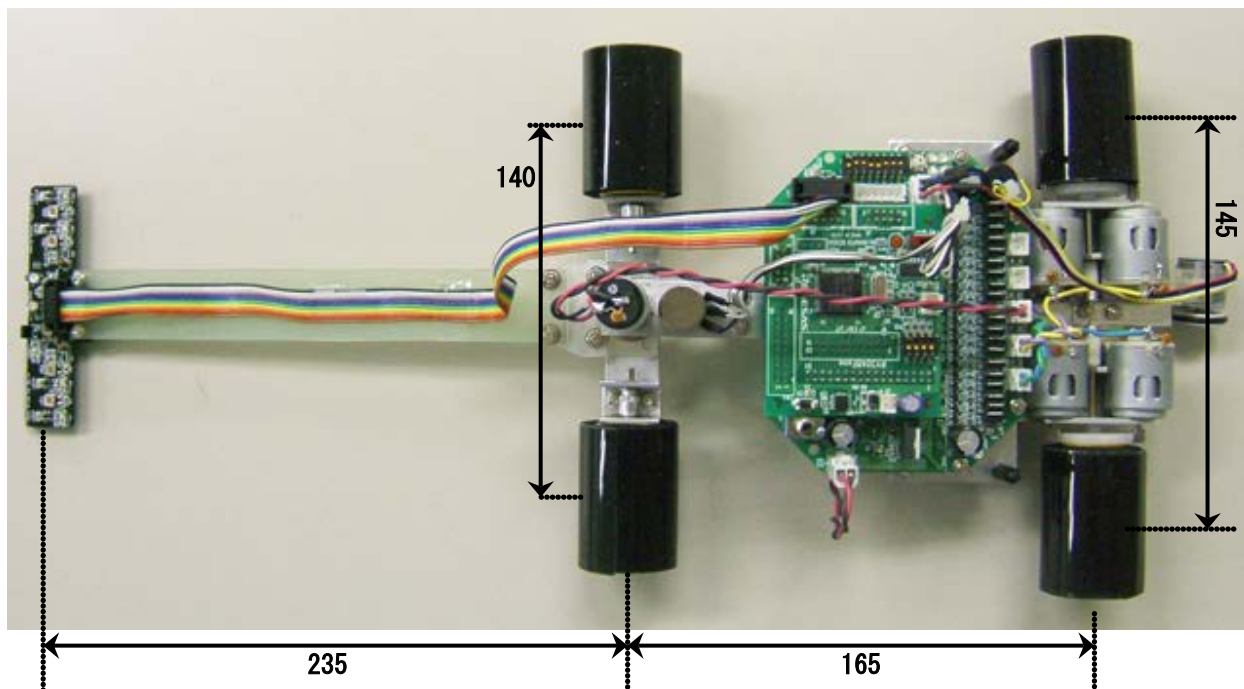
モータドライブ基板 TypeS Ver.3 には、プッシュスイッチが 1 個あります。スイッチを押すと、GND を通して"0"が、離れていると 10kΩ のプルアップ抵抗を通して"1"が入力されます。



4. 説明用マイコンカーの仕様

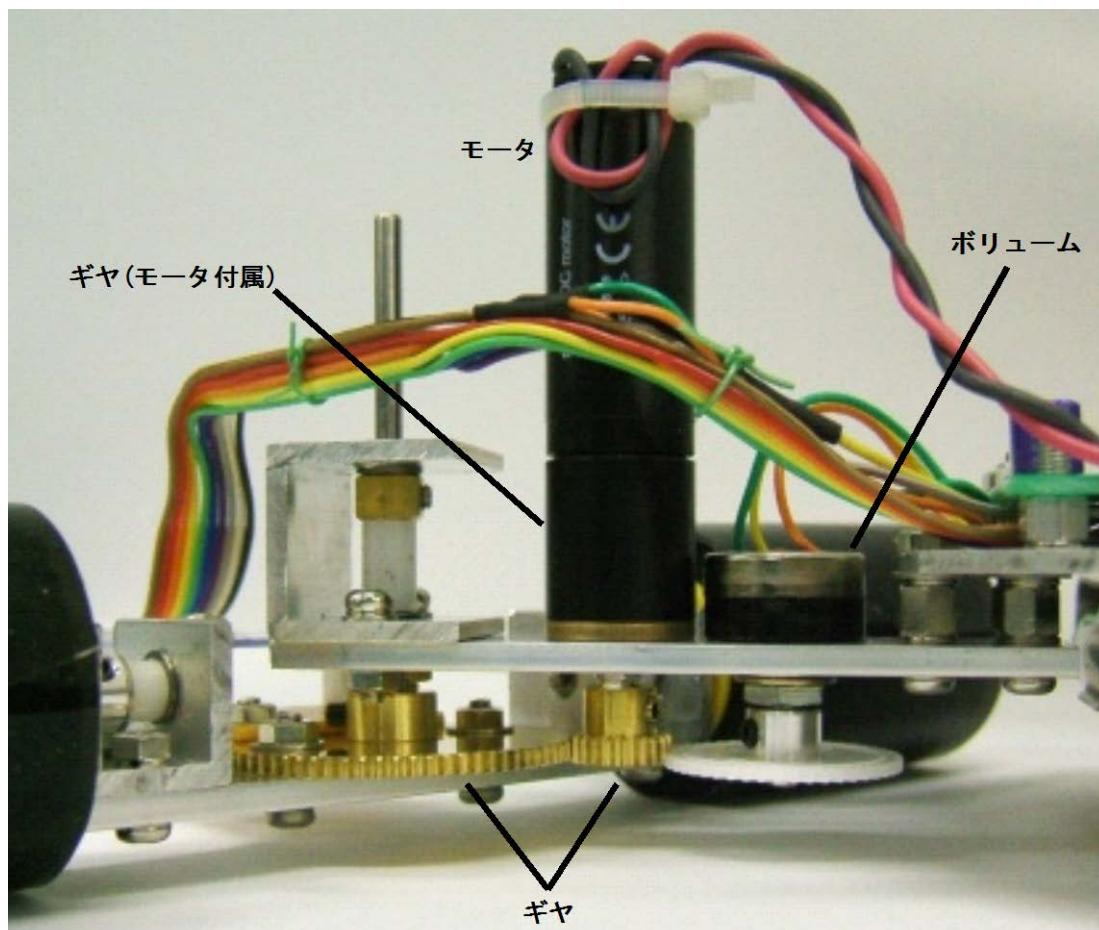
4.1 寸法

本マニュアルで説明しているマイコンカーの寸法を、下写真に示します。



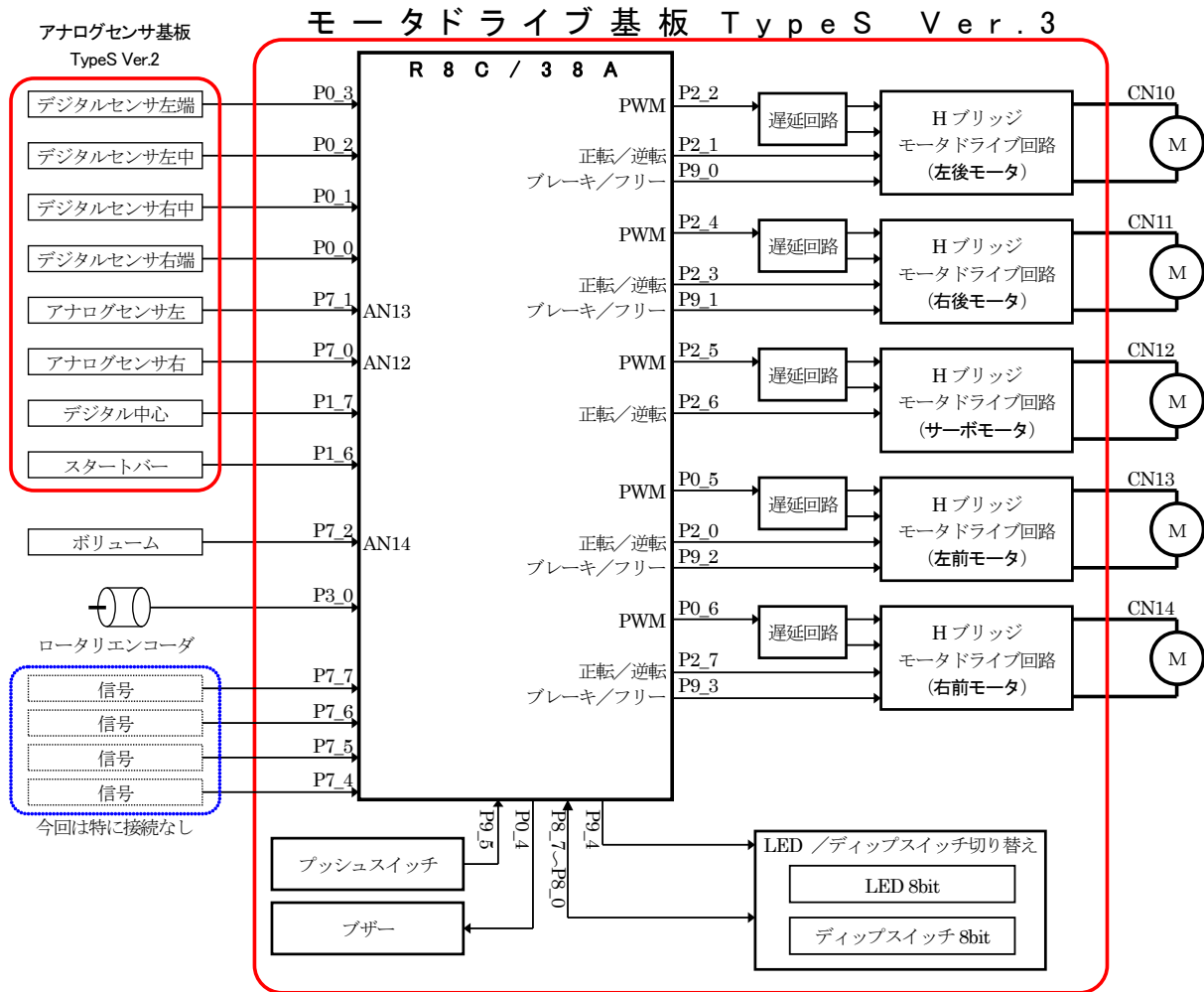
4.2 サーボ機構の自作

サーボ機構=モータドライブ回路+モータ+ギヤ+ボリューム(+制御プログラム)となります。それらをマイコンカーに組み込めば自作サーボの完成です。例として下写真に説明用マイコンカーのサーボ機構の製作例を示します。



項目	詳細
モータ	説明用マイコンカーは、マクソンモータ 118682 (RE16 3.2W)を使用しています。ギヤは110322 (GP16A 19:1)を使用しています。 最初は高性能モータで実験し、調整のコツをつかんでから安価なモータを使用すると良いでしょう。 ちなみに、指定モータ RC-260RA18130 を2個並列に接続すれば、市販されているラジコンサーボにも負けない性能にすることもできます。
ギヤ	ギヤ比はモータのトルクによりますが、今回のモータの場合、40～80 くらいが良いでしょう。写真の例では、 モータ付属のギヤ部分 $1/19 \times$ 自作部分 $20/80 = 1:76.0$ です。 ギヤの組み合わせが多すぎるとギヤの遊び(バックラッシュ)が大きくなり微妙な制御ができません。ステアリング用のギヤは、遊びを極力少なくしてください。
ボリューム	ハンドルの切れ角を検出するためのボリュームです。ハンドルをまっすぐにしたときをボリュームの中心(約 2.5V)に合わせます。左右に目一杯切ったときにボリュームの電圧が 0.5V～4.5V くらいになると良好です。それ以下の電圧でも検出範囲が狭くなるだけで問題はありません。

4.3 ブロック図



4.4 R8C/38A マイコンで使用する内蔵周辺機能

機能	詳細
A/D 変換器	P7_3～P7_0 をアナログ電圧入力用として使用します。P7_3～P7_0 端子の電圧を A/D 変換器でデジタル値に変換します。 P7_3…未接続(プルアップ抵抗 10kΩを接続) P7_2…ボリューム電圧入力 P7_1…左アナログセンサ電圧入力 P7_0…右アナログセンサ電圧入力
タイマ RB	インターバルタイマとして使用して、1ms ごとに割り込みを発生させます。
タイマ RC	PWM モードとして使用しています。タイマ RC は PWM 信号を 3 本出力することができます。タイマ RC でスピード制御しているモータを、下記に示します。 ①左前モータ ②右前モータ
タイマ RD	チャンネル 0 とチャンネル 1 を組み合わせて、リセット同期 PWM モードとして使用します。リセット同期 PWM モードは、PWM 信号を 3 本出力することができます。PWM 波形の周期は、3 本とも同じです。タイマ RD でスピード制御しているモータを、下記に示します。 ①左後モータ ②右後モータ ③自作サーボモータ
タイマ RG	ロータリエンコーダのパルス入力として使用します。


5. ワークスペース「anaservo_ver3_38a」

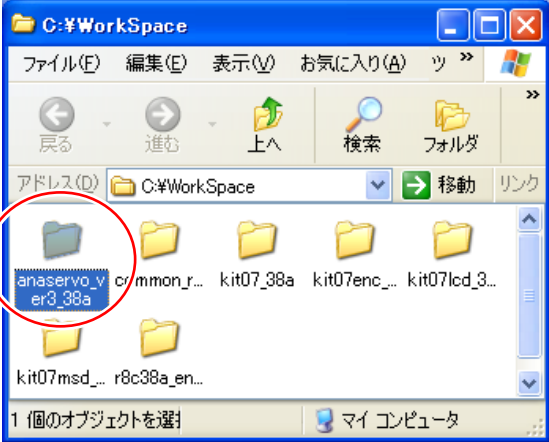
5.1 ワークスペースのインストール

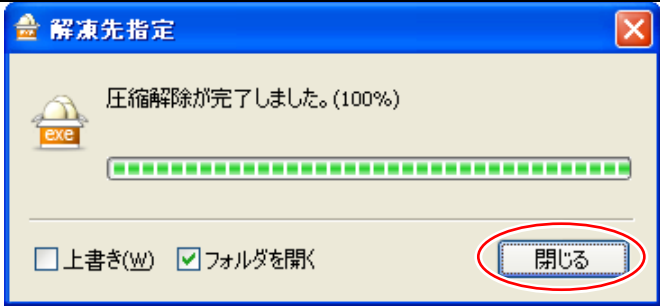
1	<p>マイコンカーラリーとは? 今から始めるマイコンカーラリー 技術情報 会記録 MCRファン倶楽部 お問い合わせ</p> <p>ダウンロード</p> <p>1/15 遅くなりましたが、2011年度の地区大会日程を掲載しました。下記の「2012年大会日程」をご覧ください。 JMCR2012大会へのホームページリニューアルは7月を予定しています。もうしばらくお待ちください。 /04「お知らせ」ページにJMCR2012についてを掲載しました。 /18「ダウンロード」ページにマイコンカーキットVer.5に関する「マイコンカーキットVer.5 本体組み立て製作マニュアル 第</p>	<p>マイコンカーラリーホームページ http://www.mcr.gr.jp/index2.html にアクセスします。 「技術情報→ダウンロード」をクリックします。</p>
---	--	--

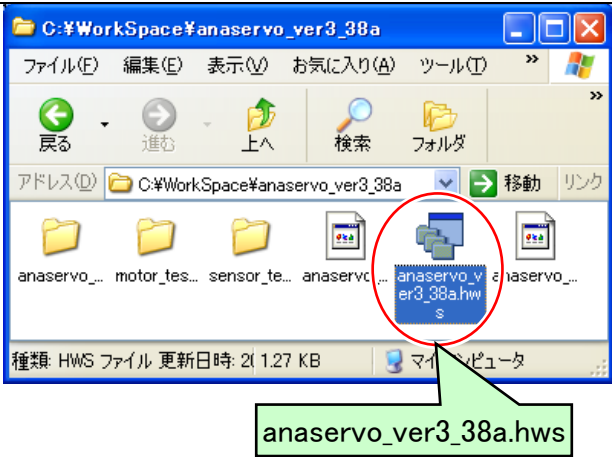
2	<p>免責事項</p> <p>「マニュアル」、「ソフトウェア」は万全な体制で制作されており、通常の使用環境においては正常に動作するように作成されていますが、万が一「マニュアル」、「ソフトウェア」による損失・損害が発生した時には、『ジャパンマイコンカーラリー実行委員会』はいかなる場合も責任を負いません。個人の免責が取れる範囲内であらかじめ了承した上でご使用くださるようお願いをいたします。</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">対象マイコン</th> <th style="text-align: center;">内容</th> <th style="text-align: center;">更新日</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">R8C/38A</td> <td>R8C/38Aマイコン(RY_R8C38ボード)に関する資料</td> <td style="text-align: center;">2013.06.03 NEW!!</td> </tr> <tr> <td style="text-align: center;">H8/3048F-ONE</td> <td>H8/3048F-ONEマイコン(RY3048Foneボード)用のサンプルプログラム、書き込みソフト</td> <td style="text-align: center;">2010.10.07</td> </tr> <tr> <td style="text-align: center;">H8/3048F</td> <td>H8/3048F-ONEマイコン(RY3048Foneボード)に</td> <td style="text-align: center;">2010.09.07</td> </tr> </tbody> </table>	対象マイコン	内容	更新日	R8C/38A	R8C/38Aマイコン(RY_R8C38ボード)に関する資料	2013.06.03 NEW!!	H8/3048F-ONE	H8/3048F-ONEマイコン(RY3048Foneボード)用のサンプルプログラム、書き込みソフト	2010.10.07	H8/3048F	H8/3048F-ONEマイコン(RY3048Foneボード)に	2010.09.07	<p>「R8C/38A マイコン (RY_R8C38 ボード)に関する資料」をクリックします。</p>
対象マイコン	内容	更新日												
R8C/38A	R8C/38Aマイコン(RY_R8C38ボード)に関する資料	2013.06.03 NEW!!												
H8/3048F-ONE	H8/3048F-ONEマイコン(RY3048Foneボード)用のサンプルプログラム、書き込みソフト	2010.10.07												
H8/3048F	H8/3048F-ONEマイコン(RY3048Foneボード)に	2010.09.07												

3	<p>ほとんどのマイコンボードで、プログラムについては、H8/3048F-ONEマイコン版の「データ解析実習マニュアル」を参照してください。</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;"> <p>モータドライブ基板TypeS Ver.3 モータ5個(駆動モータ4個、自作サーボモータ1個を想定)を制御することのできる基板です。 ※抵抗内蔵トランジスタの型式が、ロットによって変わります。性能は、一切変わりません。(2012.03.09)</p> </td> <td style="width: 30%;"> <p>モータドライブ基板TypeS Ver.3 製作マニュアル 第1.02版 2012.03.09</p> </td> <td style="width: 30%;"> <p>モータドライブ基板TypeS Ver.3 アナログセンサ基板TypeS Ver.2 プログラム解説マニュアル 第2.13版 2012.10.24</p> </td> <td style="width: 10%; text-align: center;"> <p>anaservo_ver3_38a.exe</p> </td> </tr> <tr> <td> <p>モータドライブ基板TypeS Ver.4 モータ5個(駆動モータ4個、白</p> </td> <td> <p>モータドライブ</p> </td> <td> <p>モータドライブ基板TypeS Ver.4 アナログセ</p> </td> <td></td> </tr> </table>	<p>モータドライブ基板TypeS Ver.3 モータ5個(駆動モータ4個、自作サーボモータ1個を想定)を制御することのできる基板です。 ※抵抗内蔵トランジスタの型式が、ロットによって変わります。性能は、一切変わりません。(2012.03.09)</p>	<p>モータドライブ基板TypeS Ver.3 製作マニュアル 第1.02版 2012.03.09</p>	<p>モータドライブ基板TypeS Ver.3 アナログセンサ基板TypeS Ver.2 プログラム解説マニュアル 第2.13版 2012.10.24</p>	<p>anaservo_ver3_38a.exe</p>	<p>モータドライブ基板TypeS Ver.4 モータ5個(駆動モータ4個、白</p>	<p>モータドライブ</p>	<p>モータドライブ基板TypeS Ver.4 アナログセ</p>		<p>「anaservo_ver3_38a.exe」をダウンロードします。</p>
<p>モータドライブ基板TypeS Ver.3 モータ5個(駆動モータ4個、自作サーボモータ1個を想定)を制御することのできる基板です。 ※抵抗内蔵トランジスタの型式が、ロットによって変わります。性能は、一切変わりません。(2012.03.09)</p>	<p>モータドライブ基板TypeS Ver.3 製作マニュアル 第1.02版 2012.03.09</p>	<p>モータドライブ基板TypeS Ver.3 アナログセンサ基板TypeS Ver.2 プログラム解説マニュアル 第2.13版 2012.10.24</p>	<p>anaservo_ver3_38a.exe</p>							
<p>モータドライブ基板TypeS Ver.4 モータ5個(駆動モータ4個、白</p>	<p>モータドライブ</p>	<p>モータドライブ基板TypeS Ver.4 アナログセ</p>								

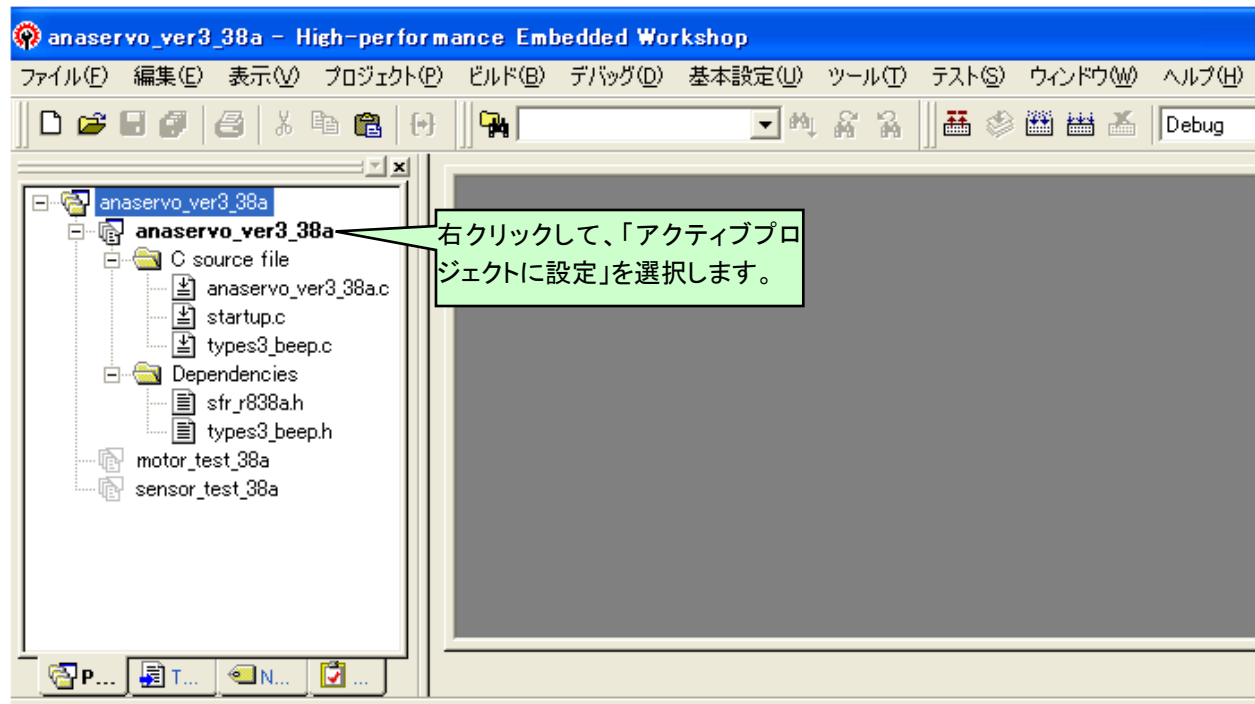
4		<p>「圧縮解除」をクリックします。</p> <p>※フォルダは変更できません。変更した場合は、ルネサス統合開発環境の設定を変更する場合がございます。</p>
---	---	---

5		<p>解凍が終わったら、自動的に「Cドライブ→Workspace」フォルダが開かれます。今回使用するのは、「anaservo_ver3_38a」です。</p>
---	---	---

6		<p>「閉じる」をクリックして終了です。</p>
---	---	--------------------------

7		<p>「Cドライブ→Workspace→anaservo_ver3_38a→anaservo_ver3_38a.hws」をダブルクリックすると、ルネサス統合開発環境が立ち上がります。</p>
---	---	---

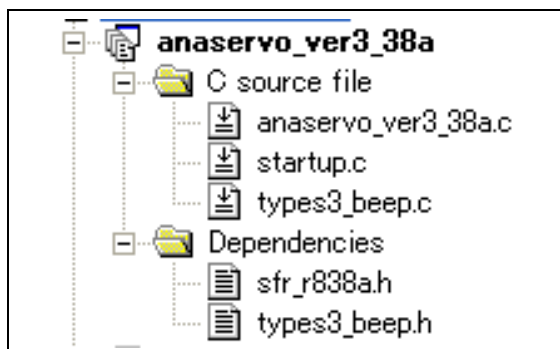
5.2 プロジェクト



ワークスペース「anaservo_ver3_38a」には、3つのプロジェクトが登録されています。

プロジェクト名	内容
anaservo_ver3_38a	モータドライブ基板 TypeS Ver.3、アナログセンサ基板 TypeS Ver.2 のそれぞれの基板を使った、アナログセンサ、自作サーボを搭載したマイコンカーの制御プログラムです。本プログラムは基本的な考え方のみ記述しています。実際にコースを完走させるには、各自プログラムを改造して対応してください。 今回は、このプロジェクトを使います。「anaservo_ver3_38a」プロジェクトをアクティブ(操作対象)にしてください。
motor_test_38a	モータドライブ基板 TypeS Ver.3 の動作テスト用プログラムです。
sensor_test_38a	アナログセンサ基板 TypeS Ver.2 の動作テスト用プログラムです。

5.3 プロジェクトの構成



	ファイル名	内容
1	anaservo_ver3_38a.c	実際に制御するプログラムが書かれています。R8C/38A の内蔵周辺機能(SFR)の初期化も行います。 ファイルの位置→C:\¥Workspace¥anaservo_ver3_38a¥anaservo_ver3_38a¥anaservo_ver3_38a.c
2	startup.c	固定割り込みベクタアドレスの設定、スタートアッププログラム、RAMの初期化(初期値のないグローバル変数、初期値のあるグローバル変数の設定)などを行います。 ファイルの位置→C:\¥Workspace¥anaservo_ver3_38a¥anaservo_ver3_38a¥startup.c
3	types3_beep.c	ブザーを制御するプログラムが書かれています。 ファイルの位置→C:\¥Workspace¥anaservo_ver3_38a¥anaservo_ver3_38a¥types3_beep.c
4	sfr_r838a.h	R8C/38A マイコンの内蔵周辺機能を制御するためのレジスタ(Special Function Registers)を定義したファイルです。 ファイルの位置→C:\¥Workspace¥common_r8c38a¥sfr_r838a.h
5	types3_beep.h	types3_beep.c のヘッダファイルです。 ファイルの位置→C:\¥Workspace¥anaservo_ver3_38a¥anaservo_ver3_38a¥types3_beep.h

6. マイコンカー走行プログラムの解説

6.1 プログラムリスト「anaservo_ver3_38a.c」

```

1 : /******
2 : /* 対象マイコン R8C/38A */
3 : /* ファイル内容 モータドライブ基板TypeS Ver. 3・アナログセンサ基板TypeS Ver. 2*/
4 : /* を使用したマイコンカートレースプログラム */
5 : /* バージョン Ver. 1.00 */
6 : /* Date 2011.06.01 */
7 : /* Copyright ジャパンマイコンカーラリー実行委員会 */
8 : /******
9 :
10 : /*
11 : 本プログラムは、
12 : ●モータドライブ基板TypeS Ver. 3
13 : ●アナログセンサ基板TypeS Ver. 2
14 : を使用したマイコンカーを動作させるプログラムです。
15 : */
16 :
17 : /*=====*/
18 : /* インクルード */
19 : /*=====*/
20 : #include <stdio.h>
21 : #include "sfr_r838a.h" /* R8C/38A SFRの定義ファイル */
22 : #include "types3_beep.h" /* ブザー追加 */
23 :
24 : /*=====*/
25 : /* シンボル定義 */
26 : /*=====*/
27 : /* 定数設定 */
28 : #define TRC_MOTOR_CYCLE 20000 /* 左前,右前モータPWMの周期 */
29 : /* 50[ns] * 20000 = 1.00[ms] */
30 : #define TRD_MOTOR_CYCLE 20000 /* 左後,右後,サホモータPWMの周期 */
31 : /* 50[ns] * 20000 = 1.00[ms] */
32 : #define FREE 1 /* モータモード フリー */
33 : #define BRAKE 0 /* モータモード ブレーキ */
34 :
35 : /*=====*/
36 : /* プロトタイプ宣言 */
37 : /*=====*/
38 : void init( void );
39 : unsigned char sensor_inp( void );
40 : unsigned char center_inp( void );
41 : unsigned char startbar_get( void );
42 : unsigned char dipsw_get( void );
43 : unsigned char dipsw_get2( void );
44 : unsigned char pushsw_get( void );
45 : unsigned char cn6_get( void );
46 : void led_out( unsigned char led );
47 : void motor_r( int accele_l, int accele_r );
48 : void motor2_r( int accele_l, int accele_r );
49 : void motor_f( int accele_l, int accele_r );
50 : void motor2_f( int accele_l, int accele_r );
51 : void motor_mode_r( int mode_l, int mode_r );
52 : void motor_mode_f( int mode_l, int mode_r );
53 : void servoPwmOut( int pwm );
54 : int check_crossline( void );
55 : int getServoAngle( void );
56 : int getAnalogSensor( void );
57 : void servoControl( void );
58 : int diff( int pwm );
59 :
60 : /*=====*/
61 : /* グローバル変数の宣言 */
62 : /*=====*/
63 : int pattern; /* マイコンカー動作パターン */
64 : int crank_mode; /* 1:クランクモード 0:通常 */
65 : unsigned long cnt1; /* タイマ用 */
66 :
67 : /* エンコーダ関連 */
68 : int iTimer10; /* 10msカウント用 */
69 : long lEncoderTotal; /* 積算値保存用 */
70 : int iEncoder; /* 10ms毎の最新値 */
71 : unsigned int uEncoderBuff; /* 計算用 割り込み内で使用 */
72 :
73 : /* サーボ関連 */
74 : int iSensorBefore; /* 前回のセンサ値保存 */
75 : int iServoPwm; /* サーボPWM値 */
76 : int iAngle0; /* 中心時のA/D値保存 */
77 :
78 : /* センサ関連 */
79 : int iSensorPattern; /* センサ状態保持用 */
80 :

```



```

81 : /* TRCレジスタのバッファ */
82 : unsigned int   trcgrb_buff;           /* TRCGRBのバッファ */
83 : unsigned int   trcgrd_buff;           /* TRCGRDのバッファ */
84 :
85 : /* モータドライブ基板TypeS Ver. 3上のLED、ディップスイッチ制御 */
86 : unsigned char   types_led;            /* LED値設定 */
87 : unsigned char   types_dipsw;          /* ディップスイッチ値保存 */
88 :
89 : /* 内輪差値計算用 各マイコンカーに合わせて再計算して下さい */
90 : const int revolution_difference[] = { /* 角度から内輪、外輪回転差計算 */
91 :     100, 98, 97, 95, 94,
92 :     92, 91, 89, 88, 87,
93 :     85, 84, 82, 81, 80,
94 :     78, 77, 76, 74, 73,
95 :     72, 70, 69, 68, 66,
96 :     65, 64, 62, 61, 60,
97 :     58, 57, 56, 54, 53,
98 :     52, 50, 49, 48, 46,
99 :     45, 43, 42, 40, 39,
100 :     38 };
101 :
102 : /*****
103 : /* メインプログラム */
104 : *****/
105 : void main( void )
106 : {
107 :     int i;
108 :
109 :     /* マイコン機能の初期化 */
110 :     init();                               /* 初期化 */
111 :     asm(" fset I ");                       /* 全体の割り込み許可 */
112 :     initBeepS();                            /* ブザー関連処理 */
113 :
114 :     /* マイコンカーの状態初期化 */
115 :     motor_mode_f( BRAKE, BRAKE );
116 :     motor_mode_r( BRAKE, BRAKE );
117 :     motor_f( 0, 0 );
118 :     motor_r( 0, 0 );
119 :     servoPwmOut( 0 );
120 :     setBeepPatternS( 0x8000 );
121 :
122 :     while( 1 ) {
123 :
124 :         switch( pattern ) {
125 :         case 0:
126 :             /* プッシュスイッチ押下待ち */
127 :             servoPwmOut( 0 );
128 :             if( pushsw_get() ) {
129 :                 setBeepPatternS( 0xcc00 );
130 :                 cnt1 = 0;
131 :                 pattern = 1;
132 :                 break;
133 :             }
134 :             i = (cnt1/200) % 2 + 1;
135 :             if( startbar_get() ) {
136 :                 i += ((cnt1/100) % 2 + 1) << 2;
137 :             }
138 :             led_out( i );                   /* LED点滅処理 */
139 :             break;
140 :
141 :         case 1:
142 :             /* スタートバー開待ち */
143 :             servoPwmOut( iServoPwm / 2 );
144 :             if( !startbar_get() ) {
145 :                 iAngle0 = getServoAngle(); /* 0度の位置記憶 */
146 :                 led_out( 0x0 );
147 :                 cnt1 = 0;
148 :                 pattern = 11;
149 :                 break;
150 :             }
151 :             led_out( 1 << (cnt1/50) % 4 );
152 :             break;
153 :
154 :         case 11:
155 :             /* 通常トレース */
156 :             servoPwmOut( iServoPwm );
157 :             i = getServoAngle();
158 :             if( i > 170 ) {
159 :                 motor_f( 0, 0 );
160 :                 motor_r( 0, 0 );
161 :             } else if( i > 25 ) {
162 :                 motor_f( diff(80), 80 );
163 :                 motor_r( diff(80), 80 );
164 :             } else if( i < -170 ) {
165 :                 motor_f( 0, 0 );
166 :                 motor_r( 0, 0 );
167 :             } else if( i < -25 ) {
168 :                 motor_f( 80, diff(80) );
169 :                 motor_r( 80, diff(80) );
170 :             } else {

```

6. マイコンカー走行プログラムの解説

```

171 :         motor_f( 100, 100 );
172 :         motor_r( 100, 100 );
173 :     }
174 :     if( check_crossline() ) {          /* クロスラインチェック          */
175 :         cnt1 = 0;
176 :         crank_mode = 1;
177 :         pattern = 21;
178 :     }
179 :     break;
180 :
181 :     case 21:
182 :         /* クロスライン通過処理 */
183 :         servoPwmOut( iServoPwm );
184 :         led_out( 0x3 );
185 :         motor_f( 0, 0 );
186 :         motor_r( 0, 0 );
187 :         if( cnt1 >= 100 ) {
188 :             cnt1 = 0;
189 :             pattern = 22;
190 :         }
191 :         break;
192 :
193 :     case 22:
194 :         /* クロスライン後のトレース、直角検出処理 */
195 :         servoPwmOut( iServoPwm );
196 :         if( iEncoder >= 11 ) {          /* エンコーダによりスピード制御 */
197 :             motor_f( 0, 0 );
198 :             motor_r( 0, 0 );
199 :         } else {
200 :             motor2_f( 70, 70 );
201 :             motor2_r( 70, 70 );
202 :         }
203 :
204 :         if( (sensor_inp() & 0x01) == 0x01 ) { /* 右クランク?          */
205 :             led_out( 0x1 );
206 :             cnt1 = 0;
207 :             pattern = 31;
208 :             break;
209 :         }
210 :         if( (sensor_inp() & 0x08) == 0x08 ) { /* 左クランク?          */
211 :             led_out( 0x2 );
212 :             cnt1 = 0;
213 :             pattern = 41;
214 :             break;
215 :         }
216 :         break;
217 :
218 :     case 31:
219 :         /* 右クランク処理 */
220 :         servoPwmOut( 50 );              /* 振りが弱いときは大きくする */
221 :         motor_f( 60, 33 );              /* この部分は「角度計算(4WD時).xls」 */
222 :         motor_r( 49, 22 );              /* で計算 */
223 :         if( sensor_inp() == 0x04 ) {    /* 曲げ終わりチェック          */
224 :             cnt1 = 0;
225 :             iSensorPattern = 0;
226 :             crank_mode = 0;
227 :             pattern = 32;
228 :         }
229 :         break;
230 :
231 :     case 32:
232 :         /* 少し時間が経つまで待つ */
233 :         servoPwmOut( iServoPwm );
234 :         motor2_r( 80, 80 );
235 :         motor2_f( 80, 80 );
236 :         if( cnt1 >= 100 ) {
237 :             led_out( 0x0 );
238 :             pattern = 11;
239 :         }
240 :         break;
241 :
242 :     case 41:
243 :         /* 左クランク処理 */
244 :         servoPwmOut( -50 );              /* 振りが弱いときは大きくする */
245 :         motor_f( 33, 60 );              /* この部分は「角度計算(4WD時).xls」 */
246 :         motor_r( 22, 49 );              /* で計算 */
247 :         if( sensor_inp() == 0x02 ) {    /* 曲げ終わりチェック          */
248 :             cnt1 = 0;
249 :             iSensorPattern = 0;
250 :             crank_mode = 0;
251 :             pattern = 42;
252 :         }
253 :         break;
254 :

```

```

255 :     case 42:
256 :         /* 少し時間が経つまで待つ */
257 :         servoPwmOut( iServoPwm );
258 :         motor2_f( 80, 80 );
259 :         motor2_r( 80, 80 );
260 :         if( cnt1 >= 100 ) {
261 :             led_out( 0x0 );
262 :             pattern = 11;
263 :         }
264 :         break;
265 :
266 :     default:
267 :         break;
268 :     }
269 : }
270 : }
271 :
272 : /******
273 : /* R8C/38A スペシャルファンクションレジスタ (SFR) の初期化
274 : /******
275 : void init( void )
276 : {
277 :     int    i;
278 :
279 :     /* クロックをXINクロック (20MHz) に変更 */
280 :     prc0 = 1; /* プロテクト解除 */
281 :     cm13 = 1; /* P4_6, P4_7をXIN-XOUT端子にする */
282 :     cm05 = 0; /* XINクロック発振 */
283 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約10ms) */
284 :     ocd2 = 0; /* システムクロックをXINにする */
285 :     prc0 = 0; /* プロテクトON */
286 :
287 :     /* ポートの入出力設定 */
288 :
289 :     /* PWM(予備)      左前M_PWM      右前M_PWM      ブザー
290 :        センサ左端    センサ左中    センサ右中    センサ右端 */
291 :     p0 = 0x00;
292 :     prc2 = 1; /* PD0のプロテクト解除 */
293 :     pd0 = 0xf0;
294 :
295 :     /* センサ中心      スタートハ
296 :        DIPSW3          DIPSW2          RxD0          TxD0
297 :        pur0 |= 0x04;   DIPSW1          DIPSW0 */
298 :     p1 = 0x00; /* P1_3~P1_0のプルアップON */
299 :     pd1 = 0x10;
300 :
301 :     /* 右前M_方向      ステアM_方向      ステアM_PWM      右後M_PWM
302 :        右後M_方向      左後M_PWM        左後M_方向      左前M_方向 */
303 :     p2 = 0x00;
304 :     pd2 = 0xff;
305 :
306 :     /* none           none           none           none
307 :        none           none           none           エンコーダA相 */
308 :     p3 = 0x00;
309 :     pd3 = 0xfe;
310 :
311 :     /* XOUT           XIN           ボード上のLED  none
312 :        none           VREF         none           none */
313 :     p4 = 0x20; /* P4_5のLED:初期は点灯 */
314 :     pd4 = 0xb8;
315 :
316 :     /* none           none           none           none
317 :        none           none           none           none */
318 :     p5 = 0x00;
319 :     pd5 = 0xff;
320 :
321 :     /* none           none           none           none
322 :        none           none           none           none */
323 :     p6 = 0x00;
324 :     pd6 = 0xff;
325 :
326 :     /* CN6.2入力      CN6.3入力      CN6.4入力      CN6.5入力
327 :        none(アナログ 予備) 角度VR        センサ_左アナログ  センサ_右アナログ */
328 :     p7 = 0x00;
329 :     pd7 = 0x00;
330 :
331 :     /* DIPSWorLED     DIPSWorLED     DIPSWorLED     DIPSWorLED
332 :        DIPSWorLED     DIPSWorLED     DIPSWorLED     DIPSWorLED */
333 :     pur2 |= 0x03; /* P8_7~P8_0のプルアップON */
334 :     p8 = 0x00;
335 :     pd8 = 0x00;
336 :
337 :     /* -             -             プッシュスイッチ  P8制御(LEDorSW)
338 :        右前M_Free     左前M_Free     右後M_Free     左後M_Free */
339 :     p9 = 0x00;
340 :     pd9 = 0x1f;
341 :
342 :     /* タイマRBの設定 */
343 :     /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
344 :        = 1 / (20*10^6) * 200 * 100
345 :        = 0.001[s] = 1[ms]

```

6. マイコンカー走行プログラムの解説

```

346 :      */
347 :      trbmr = 0x00;          /* 動作モード、分周比設定 */
348 :      trbpre = 200-1;      /* プリスケアラレジスタ */
349 :      trbpr = 100-1;      /* プライマリレジスタ */
350 :      trbic = 0x06;        /* 割り込み優先レベル設定 */
351 :      trbcr = 0x01;        /* カウント開始 */
352 :
353 :      /* A/Dコンバータの設定 */
354 :      admod = 0x33;        /* 繰り返し掃引モードに設定 */
355 :      adinsel = 0x90;      /* 入力端子P7の4端子を選択 */
356 :      adcon1 = 0x30;      /* A/D動作可能 */
357 :      asm(" nop ");       /* φADの1サイクルウェイト入れる */
358 :      adcon0 = 0x01;      /* A/D変換スタート */
359 :
360 :      /* タイマRG タイマモード(両エッジでカウント)の設定 */
361 :      timsr = 0x40;        /* TRGCLKA端子 P3_0に割り当てる */
362 :      trgcr = 0x15;        /* TRGCLKA端子の両エッジでカウント */
363 :      trgmr = 0x80;        /* TRGのカウント開始 */
364 :
365 :      /* タイマRC PWMモード設定(左前モータ、右前モータ) */
366 :      trcpsr0 = 0x40;      /* TRCIOA, B端子の設定 */
367 :      trcpsr1 = 0x33;      /* TRCIOC, D端子の設定 */
368 :      trcmr = 0x0f;        /* PWMモード選択ビット設定 */
369 :      trcrl = 0x8e;        /* ソースカウト:f1, 初期出力の設定 */
370 :      trcrr2 = 0x00;      /* 出力レベルの設定 */
371 :      trcgra = TRC_MOTOR_CYCLE - 1; /* 周期設定 */
372 :      trcgrb = trcgrb_buff = trcgra; /* P0_5端子のON幅(左前モータ) */
373 :      trcgrc = trcgra;     /* P0_7端子のON幅(予備) */
374 :      trcgrd = trcgrd_buff = trcgra; /* P0_6端子のON幅(右前モータ) */
375 :      trcric = 0x07;      /* 割り込み優先レベル設定 */
376 :      trcier = 0x01;      /* IMIAを許可 */
377 :      trcoer = 0x01;      /* 出力端子の選択 */
378 :      trcmr |= 0x80;      /* TRCカウント開始 */
379 :
380 :      /* タイマRD リセット同期PWMモード設定(左後モータ、右後モータ、サーボモータ) */
381 :      trdpsr0 = 0x08;      /* TRDIOB0, C0, D0端子設定 */
382 :      trdpsr1 = 0x05;      /* TRDIOA1, B1, C1, D1端子設定 */
383 :      trdmr = 0xf0;        /* バッファレジスタ設定 */
384 :      trdfcr = 0x01;      /* リセット同期PWMモードに設定 */
385 :      trdcr0 = 0x20;      /* ソースカウトの選択:f1 */
386 :      trdgra0 = trdgrc0 = TRD_MOTOR_CYCLE - 1; /* 周期設定 */
387 :      trdgrb0 = trdgrd0 = 0; /* P2_2端子のON幅(左後モータ) */
388 :      trdgral = trdgrcl = 0; /* P2_4端子のON幅(右後モータ) */
389 :      trdgrbl = trdgrdl = 0; /* P2_5端子のON幅(サーボモータ) */
390 :      trdoer1 = 0xcd;     /* 出力端子の選択 */
391 :      trdstr = 0x0d;      /* TRD0カウント開始 */
392 : }
393 :
394 : /*****
395 : /* タイマRB 割り込み処理
396 : /*****
397 : #pragma interrupt /B intTRB(vect=24)
398 : void intTRB( void )
399 : {
400 :     unsigned int i;
401 :
402 :     asm(" fset I ");     /* タイマRB以上の割り込み許可 */
403 :
404 :     cnt1++;
405 :
406 :     /* サーボモータ制御 */
407 :     servoControl();
408 :
409 :     /* ブザー処理 */
410 :     beepProcessS();
411 :
412 :     /* 10回中1回実行する処理 */
413 :     iTimer10++;
414 :     switch( iTimer10 ) {
415 :     case 1:
416 :         /* エンコーダ制御 */
417 :         i = trg;
418 :         iEncoder = i - uEncoderBuff;
419 :         lEncoderTotal += iEncoder;
420 :         uEncoderBuff = i;
421 :         break;
422 :
423 :     case 2:
424 :         /* スイッチ読み込み準備 */
425 :         p9_4 = 0;        /* LED出力OFF */
426 :         pd8 = 0x00;
427 :         break;
428 :
429 :     case 3:
430 :         /* スイッチ読み込み、LED出力 */
431 :         types_dipsw = ~p8; /* ドライブ基板TypeS Ver. 3のSW読み込み */
432 :         p8 = types_led;    /* ドライブ基板TypeS Ver. 3のLEDへ出力 */
433 :         pd8 = 0xff;
434 :         p9_4 = 1;        /* LED出力ON */
435 :         break;
436 :

```

```

437 :     case 4:
438 :         break;
439 :
440 :     case 5:
441 :         break;
442 :
443 :     case 6:
444 :         break;
445 :
446 :     case 7:
447 :         break;
448 :
449 :     case 8:
450 :         break;
451 :
452 :     case 9:
453 :         break;
454 :
455 :     case 10:
456 :         /* iTimer10変数の処理 */
457 :         iTimer10 = 0;
458 :         break;
459 :     }
460 : }
461 :
462 : /*****
463 : /* タイマRC 割り込み処理 */
464 : /*****
465 : #pragma interrupt intTRC(vect=7)
466 : void intTRC( void )
467 : {
468 :     trcsr &= 0xfe;
469 :
470 :     /* タイマRC デューティ比の設定 */
471 :     trcgrb = trcgrb_buff;
472 :     trcgrd = trcgrd_buff;
473 : }
474 :
475 : /*****
476 : /* アナログセンサ基板TypeS Ver. 2のデジタルセンサ値読み込み */
477 : /* 引数 なし */
478 : /* 戻り値 左端、左中、右中、右端のデジタルセンサ 0:黒 1:白 */
479 : /*****
480 : unsigned char sensor_inp( void )
481 : {
482 :     unsigned char sensor;
483 :
484 :     sensor = ~p0 & 0x0f;
485 :
486 :     return sensor;
487 : }
488 :
489 : /*****
490 : /* アナログセンサ基板TypeS Ver. 2の中心デジタルセンサ読み込み */
491 : /* 引数 なし */
492 : /* 戻り値 中心デジタルセンサ 0:黒 1:白 */
493 : /*****
494 : unsigned char center_inp( void )
495 : {
496 :     unsigned char sensor;
497 :
498 :     sensor = ~p1_7 & 0x01;
499 :
500 :     return sensor;
501 : }
502 :
503 : /*****
504 : /* アナログセンサ基板TypeS Ver. 2のスタートバー検出センサ読み込み */
505 : /* 引数 なし */
506 : /* 戻り値 0:スタートバーなし 1:スタートバーあり */
507 : /*****
508 : unsigned char startbar_get( void )
509 : {
510 :     unsigned char sensor;
511 :
512 :     sensor = ~p1_6 & 0x01;
513 :
514 :     return sensor;
515 : }
516 :

```

6. マイコンカー走行プログラムの解説

```

517 : /*******/
518 : /* マイコンボード上のディップスイッチ値読み込み */
519 : /* 引数 なし */
520 : /* 戻り値 スイッチ値 0~15 */
521 : /*******/
522 : unsigned char dipsw_get( void )
523 : {
524 :     unsigned char sw;
525 :
526 :     sw = p1 & 0x0f;          /* P1_3~P1_0読み込み */
527 :
528 :     return sw;
529 : }
530 :
531 : /*******/
532 : /* モータドライブ基板TypeS Ver. 3上のディップスイッチ値読み込み */
533 : /* 引数 なし */
534 : /* 戻り値 スイッチ値 0~255 */
535 : /*******/
536 : unsigned char dipsw_get2( void )
537 : {
538 :     /* 実際の入力はタイマRB割り込み処理で実施 */
539 :     return types_dipsw;
540 : }
541 :
542 : /*******/
543 : /* モータドライブ基板TypeS Ver. 3上のプッシュスイッチ値読み込み */
544 : /* 引数 なし */
545 : /* 戻り値 スイッチ値 0:OFF 1:ON */
546 : /*******/
547 : unsigned char pushsw_get( void )
548 : {
549 :     unsigned char sw;
550 :
551 :     sw = ~p9_5 & 0x01;
552 :
553 :     return sw;
554 : }
555 :
556 : /*******/
557 : /* モータドライブ基板TypeS Ver. 3のCN6の状態読み込み */
558 : /* 引数 なし */
559 : /* 戻り値 0~15 */
560 : /*******/
561 : unsigned char cn6_get( void )
562 : {
563 :     unsigned char data;
564 :
565 :     data = p7 >> 4;
566 :
567 :     return data;
568 : }
569 :
570 : /*******/
571 : /* モータドライブ基板TypeSのLED制御 */
572 : /* 引数 8個のLED制御 0:OFF 1:ON */
573 : /* 戻り値 なし */
574 : /*******/
575 : void led_out( unsigned char led )
576 : {
577 :     /* 実際の出力はタイマRB割り込み処理で実施 */
578 :     types_led = led;
579 : }
580 :
581 : /*******/
582 : /* 後輪の速度制御 */
583 : /* 引数 左モータ:-100~100, 右モータ:-100~100 */
584 : /* 0で停止, 100で正転100%, -100で逆転100% */
585 : /* 戻り値 なし */
586 : /*******/
587 : void motor_r( int accele_l, int accele_r )
588 : {
589 :     int sw_data;
590 :
591 :     sw_data = dipsw_get() + 5;          /* ディップスイッチ読み込み */
592 :     accele_l = accele_l * sw_data / 20;
593 :     accele_r = accele_r * sw_data / 20;
594 :
595 :     /* 左後モータ */
596 :     if( accele_l >= 0 ) {
597 :         p2_1 = 0;
598 :         trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_l / 100;
599 :     } else {
600 :         p2_1 = 1;
601 :         trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_l ) / 100;
602 :     }
603 : }

```

```

604 :      /* 右後モータ */
605 :      if( accele_r >= 0 ) {
606 :          p2_3 = 0;
607 :          trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_r / 100;
608 :      } else {
609 :          p2_3 = 1;
610 :          trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_r ) / 100;
611 :      }
612 : }
613 :
614 : /******
615 : /* 後輪の速度制御2 デイプスイッチには関係しないmotor関数
616 : /* 引数 左モータ:-100~100 , 右モータ:-100~100
617 : /*      0で停止、100で正転100%、-100で逆転100%
618 : /* 戻り値 なし
619 : /******
620 : void motor2_r( int accele_l, int accele_r )
621 : {
622 :     /* 左後モータ */
623 :     if( accele_l >= 0 ) {
624 :         p2_1 = 0;
625 :         trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_l / 100;
626 :     } else {
627 :         p2_1 = 1;
628 :         trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_l ) / 100;
629 :     }
630 :
631 :     /* 右後モータ */
632 :     if( accele_r >= 0 ) {
633 :         p2_3 = 0;
634 :         trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_r / 100;
635 :     } else {
636 :         p2_3 = 1;
637 :         trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_r ) / 100;
638 :     }
639 : }
640 :
641 : /******
642 : /* 前輪の速度制御
643 : /* 引数 左モータ:-100~100 , 右モータ:-100~100
644 : /*      0で停止、100で正転100%、-100で逆転100%
645 : /* 戻り値 なし
646 : /******
647 : void motor_f( int accele_l, int accele_r )
648 : {
649 :     int sw_data;
650 :
651 :     sw_data = dipsw_get() + 5; /* デイプスイッチ読み込み */
652 :     accele_l = accele_l * sw_data / 20;
653 :     accele_r = accele_r * sw_data / 20;
654 :
655 :     /* 左前モータ */
656 :     if( accele_l >= 0 ) {
657 :         p2_0 = 0;
658 :     } else {
659 :         p2_0 = 1;
660 :         accele_l = -accele_l;
661 :     }
662 :     if( accele_l <= 5 ) {
663 :         trcgrb = trcgrb_buff = trcgra;
664 :     } else {
665 :         trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_l / 100;
666 :     }
667 :
668 :     /* 右前モータ */
669 :     if( accele_r >= 0 ) {
670 :         p2_7 = 0;
671 :     } else {
672 :         p2_7 = 1;
673 :         accele_r = -accele_r;
674 :     }
675 :     if( accele_r <= 5 ) {
676 :         trcgrd = trcgrd_buff = trcgra;
677 :     } else {
678 :         trcgrd_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_r / 100;
679 :     }
680 : }
681 :

```

6. マイコンカー走行プログラムの解説

```

682 : /*****/
683 : /* 前輪の速度制御2 ディップスイッチには関係しない motor関数 */
684 : /* 引数 左モータ:-100~100, 右モータ:-100~100 */
685 : /* 0で停止、100で正転100%、-100で逆転100% */
686 : /* 戻り値 なし */
687 : /*****/
688 : void motor2_f( int accele_l, int accele_r )
689 : {
690 :     /* 左前モータ */
691 :     if( accele_l >= 0 ) {
692 :         p2_0 = 0;
693 :     } else {
694 :         p2_0 = 1;
695 :         accele_l = -accele_l;
696 :     }
697 :     if( accele_l <= 5 ) {
698 :         trcgrb = trcgrb_buff = trcgra;
699 :     } else {
700 :         trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_l / 100;
701 :     }
702 :
703 :     /* 右前モータ */
704 :     if( accele_r >= 0 ) {
705 :         p2_7 = 0;
706 :     } else {
707 :         p2_7 = 1;
708 :         accele_r = -accele_r;
709 :     }
710 :     if( accele_r <= 5 ) {
711 :         trcgrd = trcgrd_buff = trcgra;
712 :     } else {
713 :         trcgrd_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_r / 100;
714 :     }
715 : }
716 :
717 : /*****/
718 : /* 後モータ停止動作 (ブレーキ、フリー) */
719 : /* 引数 左モータ:FREE or BRAKE, 右モータ:FREE or BRAKE */
720 : /* 戻り値 なし */
721 : /*****/
722 : void motor_mode_r( int mode_l, int mode_r )
723 : {
724 :     if( mode_l ) {
725 :         p9_0 = 1;
726 :     } else {
727 :         p9_0 = 0;
728 :     }
729 :     if( mode_r ) {
730 :         p9_1 = 1;
731 :     } else {
732 :         p9_1 = 0;
733 :     }
734 : }
735 :
736 : /*****/
737 : /* 前モータ停止動作 (ブレーキ、フリー) */
738 : /* 引数 左モータ:FREE or BRAKE, 右モータ:FREE or BRAKE */
739 : /* 戻り値 なし */
740 : /*****/
741 : void motor_mode_f( int mode_l, int mode_r )
742 : {
743 :     if( mode_l ) {
744 :         p9_2 = 1;
745 :     } else {
746 :         p9_2 = 0;
747 :     }
748 :     if( mode_r ) {
749 :         p9_3 = 1;
750 :     } else {
751 :         p9_3 = 0;
752 :     }
753 : }
754 :
755 : /*****/
756 : /* サーボモータ制御 */
757 : /* 引数 サーボモータPWM:-100~100 */
758 : /* 0で停止、100で正転100%、-100で逆転100% */
759 : /* 戻り値 なし */
760 : /*****/
761 : void servoPwmOut( int pwm )
762 : {
763 :     if( pwm >= 0 ) {
764 :         p2_6 = 0;
765 :         trdgrd1 = (long)( TRD_MOTOR_CYCLE - 2 ) * pwm / 100;
766 :     } else {
767 :         p2_6 = 1;
768 :         trdgrd1 = (long)( TRD_MOTOR_CYCLE- 2 ) * ( -pwm ) / 100;
769 :     }
770 : }
771 :

```



```

772 : /******
773 : /* クロスライン検出処理 */
774 : /* 引数 なし */
775 : /* 戻り値 0:クロスラインなし 1:あり */
776 : /******
777 : int check_crossline( void )
778 : {
779 :     unsigned char b;
780 :     int ret = 0;
781 :
782 :     b = sensor_inp();
783 :     if( b==0x0f || b==0x0e || b==0x0d || b==0x0b || b==0x07 ) {
784 :         ret = 1;
785 :     }
786 :     return ret;
787 : }
788 :
789 : /******
790 : /* サーボ角度取得 */
791 : /* 引数 なし */
792 : /* 戻り値 入れ替え後の値 */
793 : /******
794 : int getServoAngle( void )
795 : {
796 :     return( ad2 - iAngle0 );
797 : }
798 :
799 : /******
800 : /* アナログセンサ値取得 */
801 : /* 引数 なし */
802 : /* 戻り値 センサ値 */
803 : /******
804 : int getAnalogSensor( void )
805 : {
806 :     int ret;
807 :
808 :     ret = ad1 - ad0; /* アナログセンサ情報取得 */
809 :
810 :     if( !crank_mode ) {
811 :         /* クランクモードでなければ補正処理 */
812 :         switch( iSensorPattern ) {
813 :             case 0:
814 :                 if( sensor_inp() == 0x04 ) {
815 :                     ret = -650;
816 :                     break;
817 :                 }
818 :                 if( sensor_inp() == 0x02 ) {
819 :                     ret = 650;
820 :                     break;
821 :                 }
822 :                 if( sensor_inp() == 0x0c ) {
823 :                     ret = -700;
824 :                     iSensorPattern = 1;
825 :                     break;
826 :                 }
827 :                 if( sensor_inp() == 0x03 ) {
828 :                     ret = 700;
829 :                     iSensorPattern = 2;
830 :                     break;
831 :                 }
832 :                 break;
833 :             case 1:
834 :                 /* センサ右寄り */
835 :                 ret = -700;
836 :                 if( sensor_inp() == 0x04 ) {
837 :                     iSensorPattern = 0;
838 :                 }
839 :                 break;
840 :             case 2:
841 :                 /* センサ左寄り */
842 :                 ret = 700;
843 :                 if( sensor_inp() == 0x02 ) {
844 :                     iSensorPattern = 0;
845 :                 }
846 :                 break;
847 :             }
848 :         }
849 :     }
850 : }
851 :
852 :     return ret;
853 : }
854 :

```

6. マイコンカー走行プログラムの解説

```

855 : /******
856 : /* サーボモータ制御 */
857 : /* 引数 なし */
858 : /* 戻り値 グローバル変数 iServoPwm に代入 */
859 : /******
860 : void servoControl( void )
861 : {
862 :     int i, iRet, iP, iD;
863 :     int kp, kd;
864 :
865 :     i = getAnalogSensor();          /* センサ値取得 */
866 :     kp = dipsw_get2() & 0x0f;      /* 調整できたらP,D値は固定値に */
867 :     kd = (dipsw_get2() >> 4) * 5; /* してください */
868 :
869 :     /* サーボモータ用PWM値計算 */
870 :     iP = kp * i;                    /* 比例 */
871 :     iD = kd * (iSensorBefore - i); /* 微分(目安はPの5~10倍) */
872 :     iRet = iP - iD;
873 :     iRet /= 64;
874 :
875 :     /* PWMの上限の設定 */
876 :     if( iRet > 50 ) iRet = 50;      /* マイコンカーが安定したら */
877 :     if( iRet < -50 ) iRet = -50;   /* 上限を90くらいにしてください */
878 :     iServoPwm = iRet;
879 :
880 :     iSensorBefore = i;              /* 次回はこの値が1ms前の値となる*/
881 : }
882 :
883 : /******
884 : /* 外輪のPWMから、内輪のPWMを割り出す ハンドル角度は現在の値を使用 */
885 : /* 引数 外輪PWM */
886 : /* 戻り値 内輪PWM */
887 : /******
888 : int diff( int pwm )
889 : {
890 :     int i, ret;
891 :
892 :     i = getServoAngle() / 5;        /* 1度あたりの増分で割る */
893 :     if( i < 0 ) i = -i;
894 :     if( i > 45 ) i = 45;
895 :     ret = revolution_difference[i] * pwm / 100;
896 :
897 :     return ret;
898 : }
899 :
900 : /******
901 : /* end of file */
902 : /******

```

6.2 プログラムの解説

6.2.1 シンボル定義

```

24 : /*=====*/
25 : /* シンボル定義 */
26 : /*=====*/
27 : /* 定数設定 */
28 : #define TRC_MOTOR_CYCLE 20000 /* 左前, 右前モータPWMの周期 */
29 : /* 50[ns] * 20000 = 1.00[ms] */
30 : #define TRD_MOTOR_CYCLE 20000 /* 左後, 右後, サボモータPWMの周期 */
31 : /* 50[ns] * 20000 = 1.00[ms] */
32 : #define FREE 1 /* モータモード フリー */
33 : #define BRAKE 0 /* モータモード ブレーキ */
    
```

変数名	内容
TRC_MOTOR_CYCLE	<p>タイマ RC の PWM 波形の周期を決める値です。タイマ RC では、左前モータ、右前モータを制御しています。 タイマ RC カウントソースは 20MHz の水晶振動子を使います。周期は次のようになります。</p> $1 / (20 \times 10^6) = 50.00[\text{ns}]$ <p>今回、PWM 周期は 1ms にします。よって TRC_MOTOR_CYCLE は、次のようになります。 PWM 周期 / タイマ RC カウントソース = $(1 \times 10^{-3}) / (50 \times 10^{-9}) = 20,000$</p>
TRD_MOTOR_CYCLE	<p>タイマ RD のリセット同期 PWM モードで出力する PWM 波形の周期を決める値です。タイマ RD では、左後ろモータ、右後ろモータ、自作サーボモータを制御しています。 タイマ RD カウントソースは 20MHz の水晶振動子を使います。周期は次のようになります。</p> $1 / (20 \times 10^6) = 50.00[\text{ns}]$ <p>今回、PWM 周期は 1ms にすることとします。よって TRD_MOTOR_CYCLE は、次のようになります。 PWM 周期 / タイマ RD カウントソース = $(1 \times 10^{-3}) / (50 \times 10^{-9}) = 20,000$</p>
FREE BRAKE	<p>motor_mode_r 関数、motor_mode_f 関数で使用する定数です。 モータの停止をフリーにしたい場合は「FREE」、ブレーキにしたい場合は「BRAKE」を引数にセットします。 例) motor_mode_f(BRAKE , FREE); // 左前モータの停止はブレーキ、右前モータの停止はフリー motor_mode_r(FREE, BRAKE); // 左後モータの停止はフリー、右後モータの停止はブレーキ</p>

6. マイコンカー走行プログラムの解説

6.2.2 変数の定義

```

60 : /*=====*/
61 : /* グローバル変数の宣言 */
62 : /*=====*/
63 : int          pattern;          /* マイコンカー動作パターン */
64 : int          crank_mode;      /* 1:クランクモード 0:通常 */
65 : unsigned long cnt1;          /* タイマ用 */
66 :
67 : /* エンコーダ関連 */
68 : int          iTimer10;        /* 10msカウント用 */
69 : long         lEncoderTotal;   /* 積算値保存用 */
70 : int          iEncoder;        /* 10ms毎の最新値 */
71 : unsigned int uEncoderBuff;   /* 計算用 割り込み内で使用 */
72 :
73 : /* サーボ関連 */
74 : int          iSensorBefore;   /* 前回のセンサ値保存 */
75 : int          iServoPwm;       /* サーボ P WM値 */
76 : int          iAngle0;        /* 中心時のA/D値保存 */
77 :
78 : /* センサ関連 */
79 : int          iSensorPattern;  /* センサ状態保持用 */
80 :
81 : /* TRCレジスタのバッファ */
82 : unsigned int trcgrb_buff;     /* TRCGRBのバッファ */
83 : unsigned int trcgrd_buff;     /* TRCGRDのバッファ */
84 :
85 : /* モータドライブ基板TypeS Ver.3上のLED、ディップスイッチ制御 */
86 : unsigned char types_led;      /* LED値設定 */
87 : unsigned char types_dipsw;    /* ディップスイッチ値保存 */
    
```

変数名	内容
pattern	マイコンカーの現在の動作パターンを設定します。
crank_mode	アナログセンサ値をデジタルセンサを使って補正するかしないかを設定します。 0:補正 ON(通常トレース状態) 1:補正 OFF(クロスラインを検出後とクランクトレースモード時など)
cnt1	タイマです。1msごとに増加していきます。この変数を使って100ms待つなど、時間のカウントをします。
iTimer10	ロータリエンコーダ処理は、タイマRBの割り込み関数内で行います。割り込みは1msごとに発生しますが、ロータリエンコーダ処理は10msごとです。そのため、この変数を1回の割り込みごとに足していき、10になったら処理するようにすれば10msごとに処理するのと同じことになります。
lEncoderTotal	ロータリエンコーダの積算値が保存されています。スタートしてからの距離が分かります。
iEncoder	10msごとに計測したロータリエンコーダ値の最新値が保存されます。10msごとに更新されます。
uEncoderBuff	ロータリエンコーダ変数の計算用です。通常のプログラムでは使用しません。
iSensorBefore	前回のアナログセンサ値を保存します。通常のプログラムでは使用しません。

iServoPwm	タイマ RB の割り込み関数内で計算したサーボモータ用の PWM 値保存用です。
iAngle0	ステアリング角度 0 度のときのボリューム A/D 値を保存します。
iSensorPattern	アナログセンサの A/D 値を取得する getAnalogSensor 関数で使います。アナログセンサが中央ラインをはずれたときの対処用です。通常のプログラムでは使用しません。
trcgrb_buff	TRCGRB のバッファとして使います。TRCGRB の値は、P0_5 端子から出力する PWM 波形の ON 幅(左前モータ)を設定するレジスタです。 TRCGRB の値を変えるとき、直接このレジスタの値を変えるのではなく、trcgrb_buff 変数の値を変更します。タイマ RC の割り込み関数内で、trcgrb_buff 変数の値を、TRCGRB に設定します。 これは、直接 TRCGRB の値を変更すると、PWM 波形が乱れることがあるためです。詳しくは後述します。
trcgrd_buff	TRCGRD のバッファとして使います。TRCGRD の値は、P0_6 端子から出力する PWM 波形の ON 幅(右前モータ)を設定するレジスタです。 TRCGRD の値を変えるとき、直接このレジスタの値を変えるのではなく、trcgrd_buff 変数の値を変更します。タイマ RC の割り込み関数内で、trcgrd_buff 変数の値を、TRCGRD に設定します。 これは、直接 TRCGRD の値を変更すると、PWM 波形が乱れることがあるためです。詳しくは後述します。
types_led	ディップスイッチ 8bit と LED 8bit は、P8_7～P8_0 端子に接続されており兼用となっています。LED を点灯させるときは、タイマ RB の割り込み関数内で端子を出力にして、この変数の値を P8_7～P8_0 端子に出力します。
types_dipsw	ディップスイッチ 8bit と LED 8bit は、P8_7～P8_0 端子に接続されており兼用となっています。ディップスイッチの状態を入力するときは、タイマ RB の割り込み関数内で端子を入力にして、P8_7～P8_0 端子の状態を読み込み、この変数に保存します。

6.2.3 内輪差値計算用の配列追加

```

89 : /* 内輪差値計算用 各マイコンカーに合わせて再計算して下さい */
90 : const int revolution_difference[] = { /* 角度から内輪、外輪回転差計算 */
91 :     100, 98, 97, 95, 94,
92 :     92, 91, 89, 88, 87,
93 :     85, 84, 82, 81, 80,
94 :     78, 77, 76, 74, 73,
95 :     72, 70, 69, 68, 66,
96 :     65, 64, 62, 61, 60,
97 :     58, 57, 56, 54, 53,
98 :     52, 50, 49, 48, 46,
99 :     45, 43, 42, 40, 39,
100 :     38 };

```

revolution_difference という回転の差を計算した配列を追加します。この配列は const を先頭に付けています。値の変更しない変数や配列は RAM 上に配置する必要はありません。const 型修飾子を指定するとマイコンの ROM エリアに配置されます。今回の R8C/38A マイコンはプログラム ROM が 128KB、内蔵 RAM が 10KB と RAM が少ないので、RAM の有効活用を考えて const を付けました。const を取ると RAM エリアに配置されます。

revolution_difference 配列の [] 内に数字を入れると、入れた数字番目の数値が返ってきます。[] の中に入る数字を添字といいます。添字を入れたときの値を下記に示します。

6. マイコンカー走行プログラムの解説

```
revolution_difference[ 0] = 100
revolution_difference[ 1] = 98
revolution_difference[ 2] = 97
          |           |           |
revolution_difference[45] = 38
```

46 以上の値を設定してもエラーにはなりません、不定な値が返ってきます。46 以上にしないように注意する必要があります。

値の意味は、外輪の回転を 100 としたとき、添字に現在のハンドル角度を入れると内輪の回転数が返ってくるようにしています。添字が 2 のとき、97 が返ってきます。これは外輪 100、ハンドル角度が 2 度のとき、内輪の回転数は 97 ということです。ハンドル角度が 0 度～45 度のとき、内輪の値はあらかじめ計算しておきます。

今回は、マイコンカーのトレッド、ホイールベース、ハンドル角度を入力すると、内輪の PWM 値が出力されるエクセル表「**角度計算.xls**」を用意しました。そこに、ホイールベース=0.165、トレッド=0.15 と入力します。

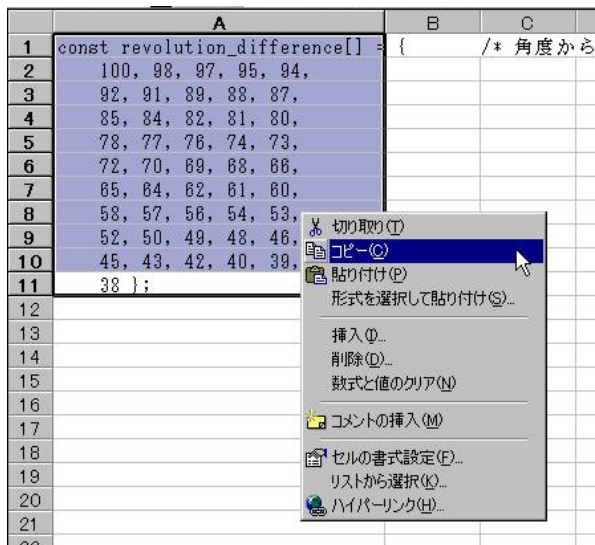
	A	B	C	D	E	F	G
1		W	0.165	m	ホイールベースを入力してください		
2		T	0.15	m	トレッドを入力してください		
3							
4		度	rad	r2	r1	r3	r1/r3*100
5		0	0				100
6		1	0.017	9.458	9.383	9.533	98
7		2	0.035	4.727	4.652	4.802	97
8		3	0.052	3.150	3.075	3.225	95
9		4	0.070	2.361	2.286	2.436	94
10		5	0.087	1.887	1.812	1.962	92
11		6	0.105	1.571	1.496	1.646	91
12		7	0.122	1.345	1.270	1.420	89
13		8	0.140	1.175	1.100	1.250	88
14		9	0.157	1.042	0.967	1.117	87
15		10	0.174	0.926	0.861	1.011	85

一覧表ができましたので、下の「コピーして貼り付け用」タグを選択、内容をコピーします。

23	18	0.314	0.508	0.433	0.583	74
24	19	0.331	0.479	0.404	0.554	73
25	20	0.349	0.454	0.379	0.529	72
26	21	0.366	0.430	0.355	0.505	70
27	22	0.384	0.409	0.334	0.484	69
28	23	0.401	0.389	0.314	0.464	68
29	24	0.419	0.371	0.296	0.446	66
30	25	0.436	0.354	0.279	0.429	65

コピーして貼り付け用

A1～A11まで選択し、右クリックで「コピー」を選択します。



プログラムの「revolution_difference」部分(89～100行)に貼り付けます。これで、自分のマイコンカーにあった内輪差が計算されました。

6.2.4 クロックの選択

R8C/38A マイコンは起動時、マイコン内蔵の低速オンチップオシレータ(約 125kHz)で動作します。これを P4_6 端子、P4_7 端子に接続されている 20MHz の水晶振動子に切り替えます。

```

272 : /*****
273 : /* R8C/38A スペシャルファンクションレジスタ (SFR) の初期化 */
274 : /*****
275 : void init( void )
276 : {
277 :     int    i;
278 :
279 :     /* クロックを XIN クロック (20MHz) に変更 */
280 :     prc0 = 1;          /* プロテクト解除 */
281 :     cm13 = 1;          /* P4_6, P4_7 を XIN-XOUT 端子にする*/
282 :     cm05 = 0;          /* XIN クロック発振 */
283 :     for(i=0; i<50; i++ ); /* 安定するまで少し待つ(約 10ms) */
284 :     ocd2 = 0;          /* システムクロックを XIN にする */
285 :     prc0 = 0;          /* プロテクト ON */
    
```

詳しくは、「マイコン実習マニュアル(R8C/38A 版)」を参照してください。

6. マイコンカー走行プログラムの解説

6.2.5 ポートの入出力設定

```

287 : /* ポートの入出力設定 */
288 :
289 : /* PWM(予備)      左前 M_PWM      右前 M_PWM      ブザー
290 :     センサ左端    センサ左中    センサ右中    センサ右端 */
291 : p0 = 0x00;
292 : prc2 = 1; /* PDO のプロテクト解除 */
293 : pd0 = 0xf0;
294 :
295 : /* センサ中心    スタートハーフ    RxD0      TxD0
296 :     DIPSW3      DIPSW2      DIPSW1      DIPSW0 */
297 : pur0 |= 0x04; /* P1_3~P1_0 のプルアップ ON */
298 : p1 = 0x00;
299 : pd1 = 0x10;
300 :
301 : /* 右前 M_方向    ステア M_方向    ステア M_PWM    右後 M_PWM
302 :     右後 M_方向    左後 M_PWM      左後 M_方向    左前 M_方向 */
303 : p2 = 0x00;
304 : pd2 = 0xff;
305 :
306 : /* none          none          none          none
307 :     none          none          none          エンコーダ A 相 */
308 : p3 = 0x00;
309 : pd3 = 0xfe;
310 :
311 : /* XOUT          XIN          ボード上の LED    none
312 :     none          VREF        none          none */
313 : p4 = 0x20; /* P4_5 の LED: 初期は点灯 */
314 : pd4 = 0xb8;
315 :
316 : /* none          none          none          none
317 :     none          none          none          none */
318 : p5 = 0x00;
319 : pd5 = 0xff;
320 :
321 : /* none          none          none          none
322 :     none          none          none          none */
323 : p6 = 0x00;
324 : pd6 = 0xff;
325 :
326 : /* CN6.2 入力    CN6.3 入力    CN6.4 入力    CN6.5 入力
327 :     none(アナログ予備) 角度 VR      センサ_左アナログ    センサ_右アナログ */
328 : p7 = 0x00;
329 : pd7 = 0x00;
330 :
331 : /* DIPSWorLED    DIPSWorLED    DIPSWorLED    DIPSWorLED
332 :     DIPSWorLED    DIPSWorLED    DIPSWorLED    DIPSWorLED */
333 : pur2 |= 0x03; /* P8_7~P8_0 のプルアップ ON */
334 : p8 = 0x00;
335 : pd8 = 0x00;
336 :
337 : /* -          -          プッシュスイッチ    P8 制御(LEDorSW)
338 :     右前 M_Free    左前 M_Free    右後 M_Free    左後 M_Free */
339 : p9 = 0x00;
340 : pd9 = 0x1f;
    
```

PD0~PD9 で、ポートの入出力設定を行います。PD0~PD9 の該当ビットを"1"にするとそのビットが出力、"0"にすると入力になります。

(1) ポートの入出力

モータドライブ基板 TypeS Ver.3、アナログセンサ基板 TypeS Ver.2 の接続機器に合わせてポートの入出力設定を行います。ポートの接続状態を、下表に示します。

ポート	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
0	未接続	左前モータ PWM 出力	右前モータ PWM 出力	ブザー 出力	デジタル センサ左端 入力	デジタル センサ左中 入力	デジタル センサ右中 入力	デジタル センサ右端 入力
1	デジタル センサ中心 入力	スタートバー 検出センサ 入力	RxD0 入力	TxD0 出力	RY_R8C38 ボード上の SW3 入力	RY_R8C38 ボード上の SW2 入力	RY_R8C38 ボード上の SW1 入力	RY_R8C38 ボード上の SW0 入力
2	右前モータ 正転/逆転 出力	サーボモータ 正転/逆転 出力	サーボモータ PWM 出力	右後モータ PWM 出力	右後モータ 正転/逆転 出力	左後モータ PWM 出力	左後モータ 正転/逆転 出力	左前モータ 正転/逆転 出力
3	未接続	未接続	未接続	未接続	未接続	未接続	未接続	ロータリ エンコーダ 入力
4	水晶振動子 (20MHz) 出力	水晶振動子 (20MHz) 入力	RY_R8C38 ボード上の LED 出力	未接続	未接続	Vcc 入力		
5	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
6	未接続	未接続	未接続	未接続	未接続	未接続	未接続	未接続
7	CN6 信号 入力	CN6 信号 入力	CN6 信号 入力	CN6 信号 入力	プルアップ抵抗 入力	ボリューム 入力	アナログ センサ左 入力	アナログ センサ右 入力
8	ディップ SW (TypeS Ver.3) または LED 出力/入力	ディップ SW (TypeS Ver.3) または LED 出力/入力	ディップ SW (TypeS Ver.3) または LED 出力/入力	ディップ SW (TypeS Ver.3) または LED 出力/入力	ディップ SW (TypeS Ver.3) または LED 出力/入力	ディップ SW (TypeS Ver.3) または LED 出力/入力	ディップ SW (TypeS Ver.3) または LED 出力/入力	ディップ SW (TypeS Ver.3) または LED 出力/入力
9			プッシュ SW 入力	ポート 8 制御 LED/ディップ SW 出力	右前モータ ブレーキ/フリー 出力	左前モータ ブレーキ/フリー 出力	右後モータ ブレーキ/フリー 出力	左後モータ ブレーキ/フリー 出力

※表の斜線の bit は、端子がない bit です。

※未接続ポートは出力設定にします。

※リセット後は、全て入力ポートです。

※P4_3、P4_4 に時計用クリスタルが接続されている場合は、両ビットとも入力にしてください。

(2) 端子のプルアップ

297 行でプルアップ制御レジスタ 0(PUR0)を設定して、P1_3~P1_0 端子のマイコン内蔵プルアップ抵抗を ON にします。

333 行でプルアップ制御レジスタ 2(PUR2)を設定して、P8_7~P8_0 端子のマイコン内蔵プルアップ抵抗を ON にします。

6. マイコンカー走行プログラムの解説

6.2.6 タイマ RB の設定

タイマ RB を使い、1ms ごとに割り込みを発生させます。

```

342 :      /* タイマ RB の設定 */
343 :      /* 割り込み周期 = 1 / 20[MHz] * (TRBPRE+1) * (TRBPR+1)
344 :                      = 1 / (20*10^6) * 200 * 100
345 :                      = 0.001[s] = 1[ms]
346 :      */
347 :      trbmr = 0x00;          /* 動作モード、分周比設定 */
348 :      trbpre = 200-1;      /* プリスケアラレジスタ */
349 :      trbpr = 100-1;      /* プライマリレジスタ */
350 :      trbic = 0x06;        /* 割り込み優先レベル設定 */
351 :      trbcr = 0x01;        /* カウント開始 */
    
```

タイマ RB に関するレジスタ設定を下記に示します。

(1) タイマ RB モードレジスタ (TRBMR: Timer RB mode register) の設定

ビット	シンボル	説明	設定値
7	tckcut_trbmr	"0"を設定	0x00
6	-	"0"を設定	
5	tck1_trbmr	タイマ RB カウントソース選択ビットを設定します。 00:f1 (1/20MHz=50ns) 01:f8 (8/20MHz=400ns)	
4	tck0_trbmr	10:タイマ RA のアンダフロー 11:f2 (2/20MHz=100ns)	
3	-	"0"を設定	
2	twrc_trbmr	"0"を設定	
1	tmod1_trbmr	タイマ RB 動作モード選択ビットを設定します。 00:タイマモード 01:プログラマブル波形発生モード	
0	tmod0_trbmr	10:プログラマブルワンショット発生モード 11:プログラマブルウェイトワンショット発生モード	

(2) タイマ RB プリスケアラレジスタ(TRBPRES:Timer RB prescaler register)の設定

(3) タイマ RB プライマリレジスタ(TRBPR:Timer RB Primary Register)の設定

ビット	シンボル	説明	設定値
TRBPRES 7~0	-	<p>割り込み周期を設定します。計算式を下記に示します。</p> $(TRBPRES+1) \times (TRBPR+1) = \text{タイマ RB 割り込み要求周期} / \text{タイマ RB カウントソース}$ <p>今回、タイマ RB 割り込み要求周期は 1ms です。タイマ RB カウントソースは、TRBMR で設定した 50ns です。よって、</p> $(TRBPRES+1) \times (TRBPR+1) = 1 \times 10^{-3} / 50 \times 10^{-9}$ $= 20,000$ <p>これを満たす TRBPRES、TRBPR を探します。今回は、 $TRBPRES+1=200$、$TRBPR=200-1$ $TRBPR+1=100$、$TRBPRES=100-1$ にします。 ただし、下記を満たすようにしてください。</p> <ul style="list-style-type: none"> •TRBPRES ≤ 255 •TRBPR ≤ 255 	200-1
TRBPR 7~0	-	上記の計算により、100-1 にします。	100-1

(4) タイマ RB 割り込み制御レジスタ(TRBIC:Timer RB interrupt control register)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x06
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	ir_trbic	"0"を設定	
2	ilvl2_trbic	他の割り込みが同時に発生した場合、どの割り込みを優先させるか設定します。レベルの高い割り込みが優先されます。割り込みを 2 つ以上使う場合は、どれを優先させるかここで決めます。今回はレベル 6 を設定します。	
1	ilvl1_trbic	000:レベル 0 (割り込み禁止)	
0	ilvl0_trbic	001:レベル 1 010:レベル 2 011:レベル 3 100:レベル 4 101:レベル 5 110:レベル 6 111:レベル 7	

6. マイコンカー走行プログラムの解説

(5) タイマ RB 制御レジスタ (TRBCR: Timer RB Control Register) の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x01
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	-	"0"を設定	
2	tstop_trbcr	"0"を設定	
1	tcstf_trbcr	"0"を設定	
0	tstart_trbcr	タイマ RB カウント開始ビットを設定します。 0:カウント停止 1:カウント開始	

6.2.7 A/D コンバータの設定

A/D コンバータを使い、アナログセンサ左、アナログセンサ右、角度検出用ボリュームの各電圧を、デジタル値に変換します。

353 :	/* A/D コンバータの設定 */		
354 :	admod = 0x33;	/* 繰り返し掃引モードに設定	*/
355 :	adinsel = 0x90;	/* 入力端子 P7 の 4 端子を選択	*/
356 :	adcon1 = 0x30;	/* A/D 動作可能	*/
357 :	asm(" nop ");	/* φAD の 1 サイクルウェイト入れる*/	
358 :	adcon0 = 0x01;	/* A/D 変換スタート	*/

A/D コンバータに関するレジスタ設定を下記に示します。

(1) A/D モードレジスタ (ADM0D: A-D mode register) の設定

ビット	シンボル	説明	設定値
7	adcap1	"0"を設定	0x33
6	adcap0	"0"を設定	
5	md2	A/D 動作モード選択を設定します。 000:単発モード 001:設定しないでください	
4	md1	010:繰り返しモード 0 011:繰り返しモード 1 100:単掃引モード	
3	md0	101:設定しないでください 110:繰り返し掃引モード 111:設定しないでください	
2	cks2	クロック源選択ビットを設定します。 0:f1 (20MHz)を選択 1:fOCO-F (高速オンチップオシレータ)を選択	
1	cks1	分周選択ビットを設定します。 00:fAD の 8 分周 (8/20MHz=400ns) 01:fAD の 4 分周 (4/20MHz=200ns) 10:fAD の 2 分周 (2/20MHz=100ns)	
0	cks0	11:fAD の 1 分周 (1/20MHz=50ns) fAD とは、bit2 で設定したクロック源のことです。このクロックを何分周で使用するか選択します。	

(2) A/D 入力選択レジスタ (ADINSEL : A-D input select register)

ビット	シンボル	説明	設定値
7	adgsel1	どのアナログ入力端子を A/D 変換するか設定します。 0000:AN0(P0_7)~AN1(P0_6)の 2 端子 0001:AN0(P0_7)~AN3(P0_4)の 4 端子	0x90
6	adgsel0	0010:AN0(P0_7)~AN5(P0_2)の 6 端子 0011:AN0(P0_7)~AN7(P0_0)の 8 端子 0100:AN8(P1_0)~AN9(P1_1)の 2 端子	
5	scan1	0101:AN8(P1_0)~AN11(P1_3)の 4 端子 1000:AN12(P7_0)~AN13(P7_1)の 2 端子 1001:AN12(P7_0)~AN15(P7_3)の 4 端子	
4	scan0	1010:AN12(P7_0)~AN17(P7_5)の 6 端子 1011:AN12(P7_0)~AN19(P7_7)の 8 端子	
3	-	"0"を設定	
2	ch2	"0"を設定	
1	ch1	"0"を設定	
0	ch0	"0"を設定	

今回は、P7_2 端子にステアリング角度検出用ボリューム、P7_1 端子にアナログセンサ左、P7_0 端子にアナログセンサ右が接続されています P7_3 はプルアップ抵抗が接続されているので 5V が常に入力されます。

(3) A/D 制御レジスタ 1 (ADCON1 : A-D control register1)

ビット	シンボル	説明	設定値
7	adddaen	"0"を設定	0x30
6	adddaen	"0"を設定	
5	adstby	A/D スタンバイビットを設定します。 0:A/D 動作停止(スタンバイ) 1:A/D 動作可能 この bit を"0"から"1"にしたときは、φ A/D の 1 サイクル以上経過した後に A/D 変換を開始します。	
4	bits	8/10 ビットモード選択ビットを設定します。 0:8 ビットモード 1:10 ビットモード	
3	-	"0"を設定	
2	-	"0"を設定	
1	-	"0"を設定	
0	adex0	"0"を設定	

(4) φ AD の 1 サイクル以上ウェイトを入れる

adstby ビットを設定した後、φ A/D の 1 サイクル以上経過した後に A/D 変換を開始しなければいけません。そのウェイトを入れるため、アセンブリ言語の nop 命令を実行します。C 言語ソースプログラム内では、アセンブリ言語は実行できないため、asm 命令というアセンブリ言語を実行できる C 言語の命令を使って nop 命令を実行します。ちなみに、nop は「no operation (何もしない)」命令で、この命令を実行するのに 1 サイクル分(50ns)の時間がかかります。

```
asm( " nop " );
```

6. マイコンカー走行プログラムの解説

(5) A/D 制御レジスタ 0(ADCON0:A-D control register0)

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x01
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	-	"0"を設定	
2	-	"0"を設定	
1	-	"0"を設定	
0	adst	A/D 変換開始フラグを設定します。 0:A/D 変換停止 1:A/D 変換開始	

6.2.8 タイマ RG の設定

タイマ RG を使い、ロータリエンコーダのパルスをカウントします。

360 :	/* タイマRG タイマモード(両エッジでカウント)の設定 */		
361 :	timsr = 0x40;	/* TRGCLKA端子 P3_0に割り当てる */	
362 :	trgcr = 0x15;	/* TRGCLKA端子の両エッジでカウント*/	
363 :	trgmr = 0x80;	/* TRGのカウント開始 */	

(1) タイマ端子選択レジスタ(TIMSR: Timer Pin Select Register)の設定

ビット	シンボル	説明	設定値
7	trgclkbsel	TRGCLKB 端子選択ビットを設定します。 0:割り当てない 1:P3_2 に割り当てる	0x40
6	trgclkasel	TRGCLKA 端子選択ビットを設定します。 0:割り当てない 1:P3_0 に割り当てる	
5	trgiobsel	"0"を設定	
4	trgioasel	"0"を設定	
3	-	"0"を設定	
2	trfisel0	"0"を設定	
1	-	"0"を設定	
0	treosel0	"0"を設定	

(2) タイマ RG 制御レジスタ(TRGCR: Timer RG Control Register)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x15
6	cclr1_trgcr	"0"を設定	
5	cclr0_trgcr	"0"を設定	
4	ckeg1_trgcr	外部クロック有効エッジ選択ビットを設定します。 00: 立ち上がりエッジでカウント 01: 立ち下がりエッジでカウント	
3	ckeg0_trgcr	10: 立ち上がり/立ち下がり両エッジでカウント 11: 設定しないでください	
2	tck2_trgcr	カウントソース選択ビットを設定します。 000: f1 (1/20MHz=50ns) 001: f2 (2/20MHz=100ns) 010: f4 (4/20MHz=200ns) 011: f8 (8/20MHz=400ns) 100: f32 (32/20MHz=1600ns)	
1	tck1_trgcr	101: TRGCLKA 入力(P3_0 端子から入力)	
0	tck0_trgcr	110: fOCO40M 111: TRGCLKB 入力(P3_2 端子から入力)	

(3) タイマ RG モードレジスタ(TRGMR: Timer RG Mode Register)の設定

ビット	シンボル	説明	設定値
7	tstart_trgmr	TRG カウント開始ビットを設定します。 0: カウント停止 1: カウント開始	0x80
6	-	"0"を設定	
5	dfck1_trgmr	"0"を設定	
4	dfck0_trgmr	"0"を設定	
3	dfb_trgmr	"0"を設定	
2	dfa_trgmr	"0"を設定	
1	mdf_trgmr	"0"を設定	
0	pwm_trgmr	"0"を設定	

6. マイコンカー走行プログラムの解説

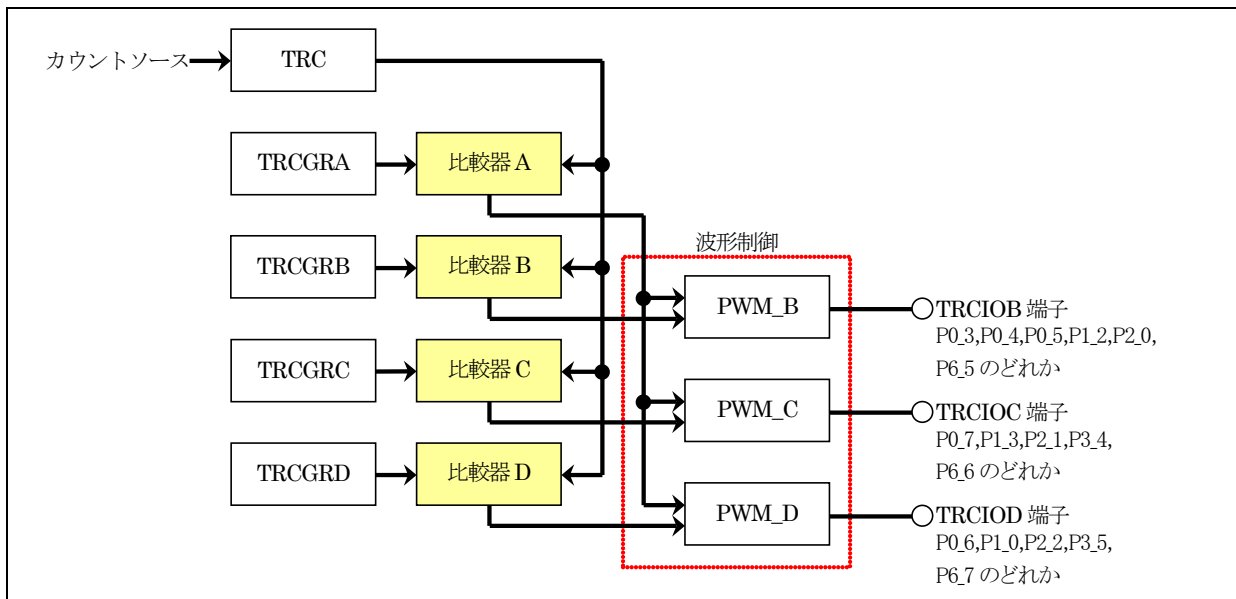
6.2.9 タイマ RC の設定

```

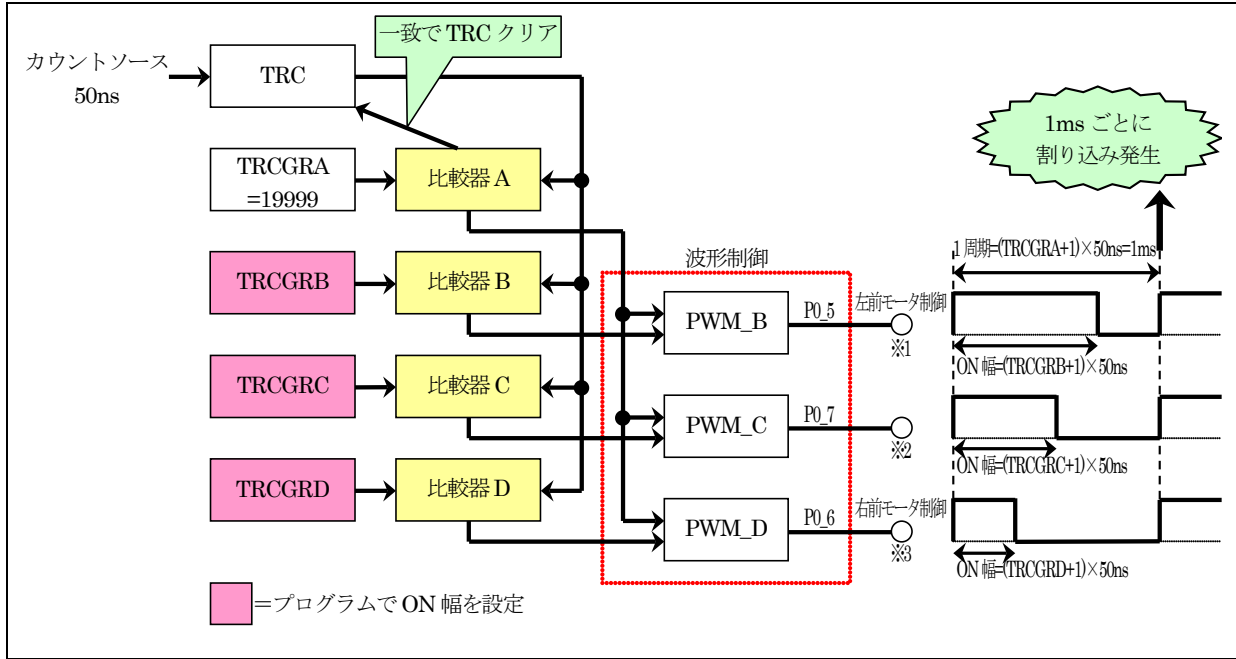
365 : /* タイマRC PWMモード設定(左前モータ、右前モータ) */
366 :   trcpsr0 = 0x40; /* TRCIOA, B端子の設定 */
367 :   trcpsr1 = 0x33; /* TRCIOC, D端子の設定 */
368 :   trcmr = 0x0f; /* PWMモード選択ビット設定 */
369 :   trccr1 = 0x8e; /* 1-カウント:f1, 初期出力の設定 */
370 :   trccr2 = 0x00; /* 出力レベルの設定 */
371 :   trcgra = TRC_MOTOR_CYCLE - 1; /* 周期設定 */
372 :   trcgrb = trcgrb_buff = trcgra; /* P0_5端子のON幅(左前モータ) */
373 :   trcgrc = trcgra; /* P0_7端子のON幅(予備) */
374 :   trcgrd = trcgrd_buff = trcgra; /* P0_6端子のON幅(右前モータ) */
375 :   trcic = 0x07; /* 割り込み優先レベル設定 */
376 :   trcier = 0x01; /* IMIAを許可 */
377 :   trcoer = 0x01; /* 出力端子の選択 */
378 :   trcmr |= 0x80; /* TRCカウント開始 */
    
```

タイマ RC を使うと、同一周期の PWM 波形を 3 本出力することができます。ただし、ON 幅を設定するタイミングによっては、波形が乱れる場合がありますので、プログラムで対処が必要です。

タイマ RC を使った PWM 波形出力の代表的な様子を、下図に示します。



今回の設定をした PWM 波形が出力される様子を、下図に示します。



- ※1…この端子は、TRCPSR0 で端子の割り当て、TRCMR で PWM モードに設定、TRCCR1 で初期出力の設定、TRCCR2 で出力レベルの設定、TRCOER で出力許可することによって、PWM 波形が出力されます。
- ※2…この端子は、TRCPSR1 で端子の割り当て、TRCMR で PWM モードに設定、TRCCR1 で初期出力の設定、TRCCR2 で出力レベルの設定、TRCOER で出力許可することによって、PWM 波形が出力されます。
- ※3…この端子は、TRCPSR1 で端子の割り当て、TRCMR で PWM モードに設定、TRCCR1 で初期出力の設定、TRCCR2 で出力レベルの設定、TRCOER で出力許可することによって、PWM 波形が出力されます。

(1) タイマ RC 端子選択レジスタ 0 (TRCPSR0: Timer RC function select register 0) の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x40
6	trciobsel2	TRCIOB 端子選択ビットを設定します。 000: TRCIOB 端子は使用しない 001: P1_2 に割り当てる	
5	trciobsel1	010: P0_3 に割り当てる 011: P0_4 に割り当てる 100: P0_5 に割り当てる	
4	trciobsel0	101: P2_0 に割り当てる 110: P6_5 に割り当てる 上記以外: 設定しないでください	
3	-	"0"を設定	
2	trcioasel2	TRCIOA/TRCTRG 端子選択ビットを設定します。 000: TRCIOA/TRCTRG 端子は使用しない	
1	trcioasel1	001: P1_1 に割り当てる 010: P0_0 に割り当てる 011: P0_1 に割り当てる	
0	trcioasel0	100: P0_2 に割り当てる 上記以外: 設定しないでください	

6. マイコンカー走行プログラムの解説

(2) タイマ RC 端子選択レジスタ 1(TRCPSR1:Timer RC function select register 1)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x33
6	trciodsel2	TRCIOD 端子選択ビットを設定します。 000:TRCIOD 端子は使用しない	
5	trciodsel1	001:P1_0 に割り当てる 010:P3_5 に割り当てる 011:P0_6 に割り当てる	
4	trciodsel0	100:P2_2 に割り当てる 101:P6_7 に割り当てる	
3	-	"0"を設定	
2	trciocsel2	TRCIOC 端子選択ビットを設定します。 000:TRCIOC 端子は使用しない	
1	trciocsel1	001:P1_3 に割り当てる 010:P3_4 に割り当てる 011:P0_7 に割り当てる	
0	trciocsel0	100:P2_1 に割り当てる 101:P6_6 に割り当てる	

(3) タイマ RC モードレジスタ(TRCMR:Timer RC mode register)の設定

ビット	シンボル	説明	設定値
7	tstart_trcmr	TRC カウント開始ビットを設定します。 0:カウント停止 1:カウント開始 カウント開始は最後にします。今回は"0"を設定します。	0x0f
6	-	"0"を設定	
5	bfd_trcmr	TRCGRD レジスタ機能選択ビットを設定します。 0:ジェネラルレジスタ 1:TRCGRB レジスタのバッファレジスタ	
4	bfc_trcmr	TRCGRC レジスタ機能選択ビットを設定します。 0:ジェネラルレジスタ 1:TRCGRA レジスタのバッファレジスタ	
3	pwm2_trcmr	PWM2 モード選択ビットを設定します。 0:PWM2 モード 1:タイマモードまたは PWM モード	
2	pwm_d_trcmr	TRCIOD PWM モード選択ビットを設定します。 0:タイマモード 1:PWM モード	
1	pwm_c_trcmr	TRCIOC PWM モード選択ビットを設定します。 0:タイマモード 1:PWM モード	
0	pwm_b_trcmr	TRCIOB PWM モード選択ビットを設定します。 0:タイマモード 1:PWM モード	

(4) タイマ RC 制御レジスタ 1(TRCCR1: Timer RC control register 1)の設定

ビット	シンボル	説明	設定値
7	cclr_trccr1	"1"を設定	0x8e
6	tck2_trccr1	カウントソース選択ビットを設定します。 000:f1 (1/20MHz=50ns)	
5	tck1_trccr1	001:f2 (2/20MHz=100ns) 010:f4 (4/20MHz=200ns) 011:f8 (8/20MHz=400ns) 100:f32 (32/20MHz=1600ns) 101:TRCCLK 入力の立ち上がりエッジ	
4	tck0_trccr1	110:fOCO40M (高速オンチップオシレータ 40MHz=今回は未接続) 111:fOCO-F (高速オンチップオシレータをFRA2 で分周したクロック=今回は未接続)	
3	tod_trccr1	TRCIOD 出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	
2	toc_trccr1	TRCIOC 出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	
1	tob_trccr1	TRCIOB 出力レベル選択ビットを設定します。 0:初期出力はアクティブでないレベル 1:初期出力はアクティブレベル	
0	toa_trccr1	"0"を設定	

(5) タイマ RC 制御レジスタ 2(TRCCR2: Timer RC control register 2)の設定

ビット	シンボル	説明	設定値
7	tceg1_trccr2	"0"を設定	0x00
6	tceg0_trccr2	"0"を設定	
5	cstp_trccr2	"0"を設定	
4	-	"0"を設定	
3	-	"0"を設定	
2	pold_trccr2	PWM モードアウトプットレベル制御ビット D を設定します。 0:TRCIOD の出力レベルは“L”アクティブ 1:TRCIOD の出力レベルは“H”アクティブ	
1	pole_trccr2	PWM モードアウトプットレベル制御ビット C を設定します。 0:TRCIOC の出力レベルは“L”アクティブ 1:TRCIOC の出力レベルは“H”アクティブ	
0	polb_trccr2	PWM モードアウトプットレベル制御ビット B を設定します。 0:TRCIOB の出力レベルは“L”アクティブ 1:TRCIOB の出力レベルは“H”アクティブ	

6. マイコンカー走行プログラムの解説

(6) タイマ RC ジェネラルレジスタ A(TRCGRA:Timer RC General register A)の設定

ビット	シンボル	説明	設定値
15~0	-	<p>TRCGRA は、PWM 波形の周期を設定します。計算式を下記に示します。</p> $\text{TRCGRA} = \text{PWM 波形の周期} / \text{タイマ RC カウンタのカウントソース} - 1$ <p>今回、タイマ RC の PWM 波形の周期は 1ms に設定します。タイマ RC カウンタのカウントソースは、TRCCR1 で設定した 50ns です。よって、</p> $\begin{aligned} \text{TRCGRA} &= (1 \times 10^{-3}) / (50 \times 10^{-9}) - 1 \\ &= 20000 - 1 \end{aligned}$ <p>となります。</p> <p>プログラムでは、define 文を使って</p> <pre>28 : #define TRC_MOTOR_CYCLE 20000 /* 左前, 右前モータ PWM の周期 */</pre> <p>とタイマ RC の PWM 波形の周期を記号定数で設定しています。よってプログラムでは、</p> <pre>371 : trcgra = TRC_MOTOR_CYCLE - 1;</pre> <p>としています。</p>	19999

(7) タイマ RC ジェネラルレジスタ B(TRCGRB:Timer RC General register B)の設定

ビット	シンボル	説明	設定値
15~0	-	<p>TRCGRB は、P0.5 端子の ON 幅を設定します。今回は、左前モータの PWM 波形の ON 幅を設定します。計算式を下記に示します。</p> $\text{TRCGRB} = \text{PWM 波形の ON 幅} / \text{タイマ RC カウンタのカウントソース} - 1$ <p>ただし、設定によっては下記のようになります。</p> <ul style="list-style-type: none"> ①TRCGRB の値を、TRCGRA と同じ値にすると ON 幅は 0%になる ②TRCGRB の値を、TRCGRA+1 より大きい値にすると ON 幅は 100%になる ③TRCGRB を設定するタイミングによっては、ON 幅 100%の波形が 1 周期出力されることがある <p>③は 1 周期ですがモータの回転が 100%になり問題です。設定するタイミングについては後述します。</p> <p>ここではまだ ON 幅は 0%にするので、TRCGRA と同じ値にします。</p>	19999

(8) タイマ RC ジェネラルレジスタ C(TRCGRC:Timer RC General register C)の設定

ビット	シンボル	説明	設定値
15~0	-	TRCGRC は、P0_7 端子の ON 幅を設定します。今回は、この端子は解放端子です。 設定内容は、TRCGRBと同じです。ON 幅は0%にすることとし、TRCGRAと同じ値にします。	19999

(9) タイマ RC ジェネラルレジスタ D(TRCGRD:Timer RC General register D)の設定

ビット	シンボル	説明	設定値
15~0	-	TRCGRD は、P0_6 端子の ON 幅を設定します。今回は、右前モータの PWM 波形の ON 幅を設定します。 設定内容は、TRCGRBと同じです。ここではまだ ON 幅は0%にするので、TRCGRAと同じ値にします。	19999

(10) タイマ RC 割り込み制御レジスタ(TRCIC:Timer RC interrupt control register)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x07
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	ir_trcic	"0"を設定	
2	ilvl2_trcic	他の割り込みが同時に発生した場合、どの割り込みを優先させるか設定します。レベルの高い割り込みが優先されます。割り込みを 2 つ以上使う場合は、どれを優先させるかここで決めます。今回はレベル 7 にします。 000:レベル 0 (割り込み禁止)	
1	ilvl1_trcic	001:レベル 1 010:レベル 2 011:レベル 3 100:レベル 4	
0	ilvl0_trcic	101:レベル 5 110:レベル 6 111:レベル 7	

6. マイコンカー走行プログラムの解説

(11) タイマ RC 割り込み許可レジスタ(TRCIER: Timer RC Interrupt Enable Register)の設定

ビット	シンボル	説明	設定値
7	ovie_trcier	オーバフロー割り込み許可ビットを設定します。 0:OVF ビットによる割り込み(OVI)禁止 1:OVF ビットによる割り込み(OVI)許可	0x01
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	imied_trcier	インプットキャプチャ/コンペア一致割り込み許可ビット D を設定します。 0:IMFD ビットによる割り込み(IMID)禁止 1:IMFD ビットによる割り込み(IMID)許可	
2	imiec_trcier	インプットキャプチャ/コンペア一致割り込み許可ビット C を設定します。 0:IMFC ビットによる割り込み(IMIC)禁止 1:IMFC ビットによる割り込み(IMIC)許可	
1	imieb_trcier	インプットキャプチャ/コンペア一致割り込み許可ビット B を設定します。 0:IMFB ビットによる割り込み(IMIB)禁止 1:IMFB ビットによる割り込み(IMIB)許可	
0	imiea_trcier	インプットキャプチャ/コンペア一致割り込み許可ビット A を設定します。 0:IMFA ビットによる割り込み(IMIB)禁止 1:IMFA ビットによる割り込み(IMIB)許可 TRC と TRCGRA の値が一致したら、割り込みを発生させる設定です。 TRCGRA は PMW 波形の周期を設定しているので、今回は 1ms ごとに割り込みが発生することになります。 タイマ RC の割り込みプログラムでの処理内容は後述します。	

(12) タイマ RC アウトプットマスタ許可レジスタ(TRCOER: Timer RC output master enable register)の設定

ビット	シンボル	説明	設定値
7	pto_trcoer	"0"を設定	0x01
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	ed_trcoer	TRCIOD 出力禁止ビットを設定します。 0:出力許可(P0_6 端子を PWM 出力にする) 1:出力禁止(TRCIOD 端子はプログラマブル入出力ポート)	
2	ec_trcoer	TRCIOC 出力禁止ビットを設定します。 0:出力許可(P0_7 端子を PWM 出力にする) 1:出力禁止(TRCIOC 端子はプログラマブル入出力ポート)	
1	eb_trcoer	TRCIOB 出力禁止ビットを設定します。 0:出力許可(P0_5 端子を PWM 出力にする) 1:出力禁止(TRCIOB 端子はプログラマブル入出力ポート)	
0	ea_trcoer	TRCIOA 出力禁止ビットを設定します。 0:出力許可 1:出力禁止(TRCIOA 端子はプログラマブル入出力ポート)	

(13) タイマ RC モードレジスタ(TRCMR: Timer RC mode register)の設定

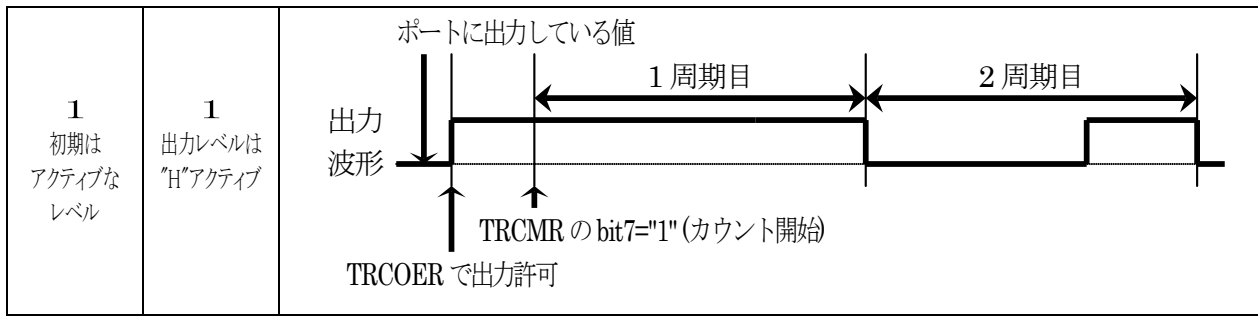
ビット	シンボル	説明	設定値
7	tstart_trcmr	TRC カウント開始ビットを設定します。 0: カウント停止 1: カウント開始	0x80 で OR
6	-	変更せず	
5	bfd_trcmr	変更せず	
4	bfc_trcmr	変更せず	
3	pwm2_trcmr	変更せず	
2	pwm_d_trcmr	変更せず	
1	pwm_c_trcmr	変更せず	
0	pwm_b_trcmr	変更せず	

(14) 出力されるタイミングと初期出力、アクティブレベルについて

タイマ RC 制御レジスタ 1 (TRCCR1) の初期出力を設定するビット (bit3~1) と、タイマ RC 制御レジスタ 2 (TRCCR2) のアクティブレベルを設定するビット (bit2~0) の関係を、下図に示します。

TRCCR1 bit3~1	TRCCR2 bit2~0	波形
0 初期は アクティブで ないレベル	0 出力レベルは "L" アクティブ	<p>ポートに出力している値</p> <p>出力波形</p> <p>1 周期目</p> <p>2 周期目</p> <p>TRCMR の bit7="1" (カウント開始)</p> <p>TRCOER で出力許可</p>
0 初期は アクティブで ないレベル	1 出力レベルは "H" アクティブ	<p>ポートに出力している値</p> <p>出力波形</p> <p>1 周期目</p> <p>2 周期目</p> <p>TRCMR の bit7="1" (カウント開始)</p> <p>TRCOER で出力許可</p>
1 初期は アクティブな レベル	0 出力レベルは "L" アクティブ	<p>ポートに出力している値</p> <p>出力波形</p> <p>1 周期目</p> <p>2 周期目</p> <p>TRCMR の bit7="1" (カウント開始)</p> <p>TRCOER で出力許可</p>

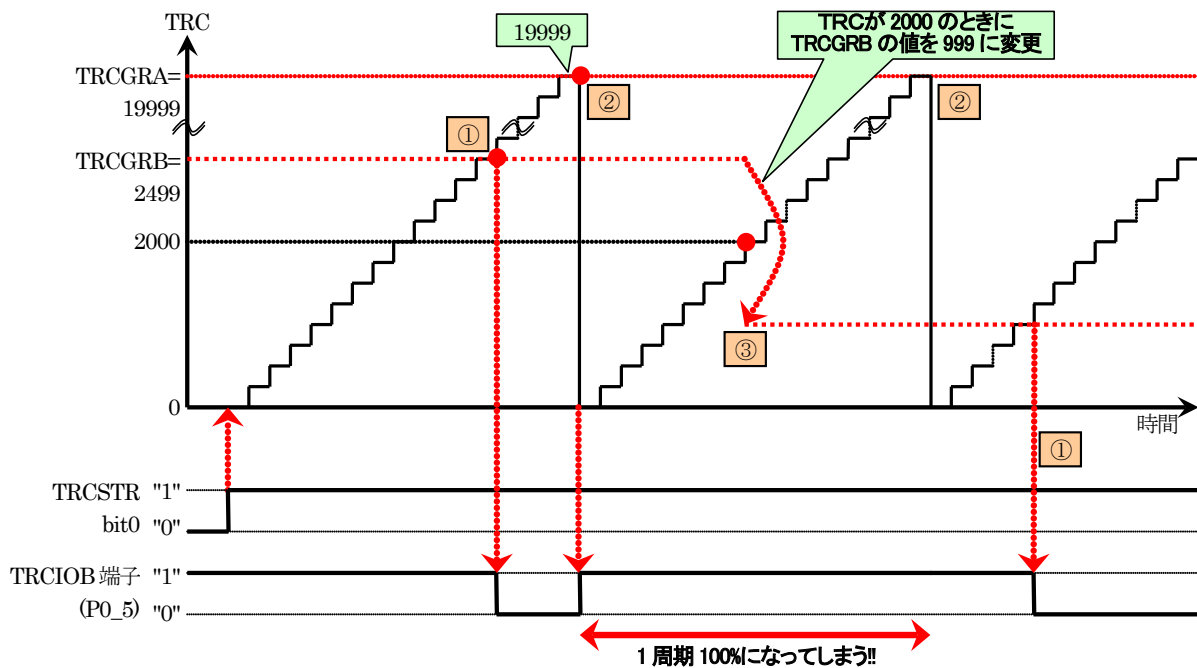
6. マイコンカー走行プログラムの解説



今回の設定は、「初期はアクティブなレベル、出力レベルは「L」アクティブ」にしています。

(15) PWM 波形

各レジスタの値と P0_5 端子の PWM 波形の様子を、下図に示します。



①	①TRC = TRCGRB+1 になった瞬間、出力波形は"0"になります。
②	②TRC = TRCGRA+1 になった瞬間、"1"になります。TRC はこのタイミングで 0 になります。PWM 波形は、下記の①と②を繰り返すことにより、PWM 波形が出力されます。
③	ON 幅を変えるときは、プログラムで TRCGRB の値を変えます。このとき、図の③のように TRCGRB の値を変えてしまった場合、①の波形が"0"になるタイミングがなく、1 周期 100%の波形が出力されてしまいます。

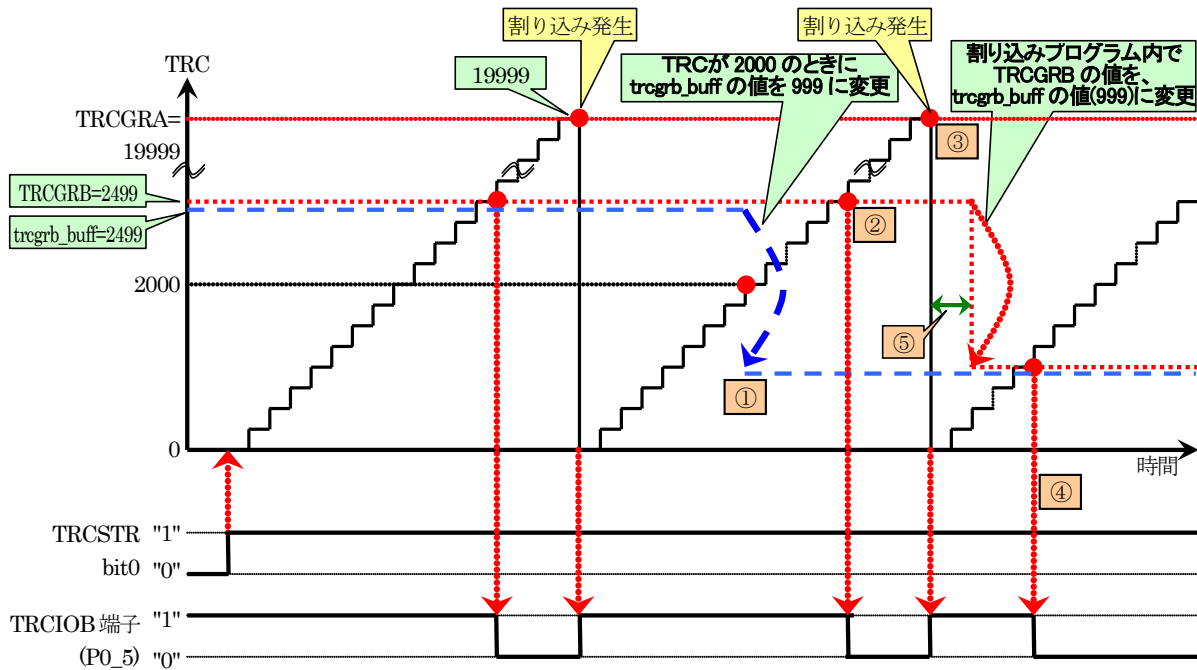
そこで、trcgrb_buffという変数を作り、プログラムではtrcgrb_buff変数の値を変更して、TRCGRBの値は直接変更しません。②のタイミングで割り込みが発生するようにして、割り込みプログラム内で trcgrb_buff 変数の値を、TRCGRB へ代入するようにします。

割り込みプログラムを、下記に示します。

```

462 : /*****/
463 : /* タイマRC 割り込み処理 */
464 : /*****/
465 : #pragma interrupt intTRC(vect=7)
466 : void intTRC( void )
467 : {
468 :     trcsr &= 0xfe;
469 :
470 :     /* タイマRC デューティ比の設定 */
471 :     trcgrb = trcgrb_buff;
472 :     trcgrd = trcgrd_buff;
473 : }
    
```

割り込みプログラムの処理を含めた P0_5 端子の PWM 波形の様子を、下図に示します。



①	プログラムでは TRCGRB の値ではなく、trcgrb_buff 変数の値を変えます。
②	TRCGRB の値は変わっていないため「TRC = TRCGRB+1」になり、波形は「0」になります。
③	タイマ RC の割り込みプログラム内で trcgrb_buff 変数を TRCGRB に代入します。
④	「TRC = TRCGRB+1」になり、波形は「0」になります。
⑤	<p>割り込みは「TRC = TRCGRB+1」になった瞬間にかかりますが、下記の理由により代入は若干遅れます。</p> <ul style="list-style-type: none"> ・現在実行している命令(アセンブリ言語レベル)が終わるまでに数百 ns～数 μs の時間がかかります。 ・割り込み関数を実行するまでに数 μs の時間がかかります。 ・プログラムを実行するのに数百 ns～数 μs の時間がかかります。 <p>よって、TRCGRB の値が小さい場合、代入した時点で TRC の値が TRCGRB より大きいことがあり、「TRC = TRCGRB+1」が起らず 1 周期 100% の PWM 波形が出力されてしまうことがあります。trcgrb_buff 変数には 1000 以上(5 μs 以上)の値を代入してください。0%にするときは、TRCGRB に TRCGRA の値を代入してください。</p>

6. マイコンカー走行プログラムの解説

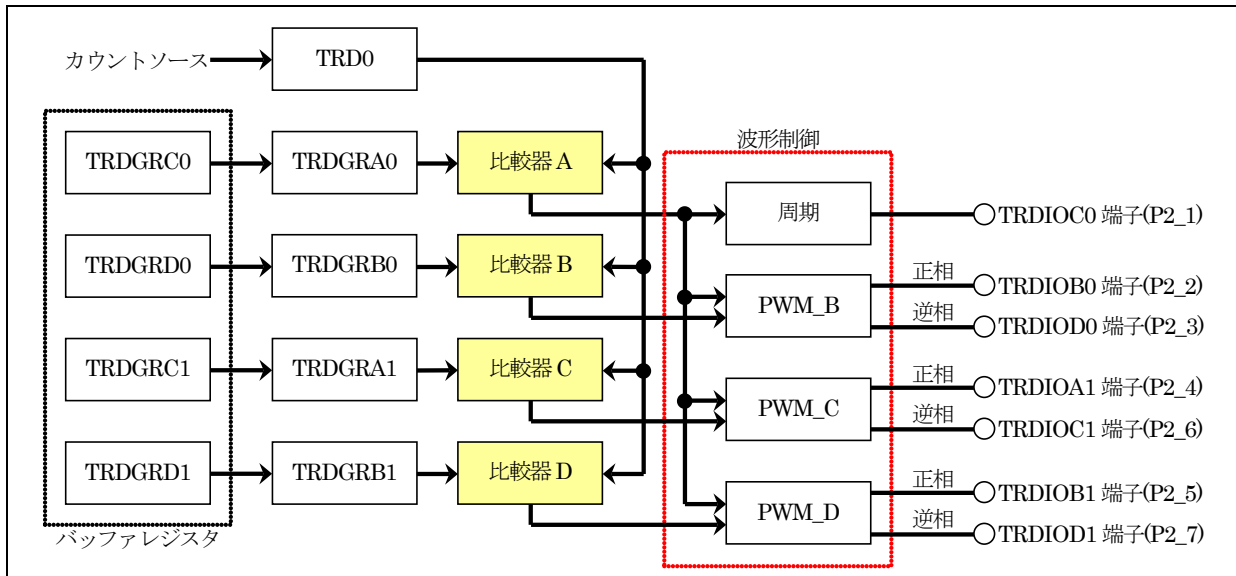
6.2.10 タイマ RD の設定

```

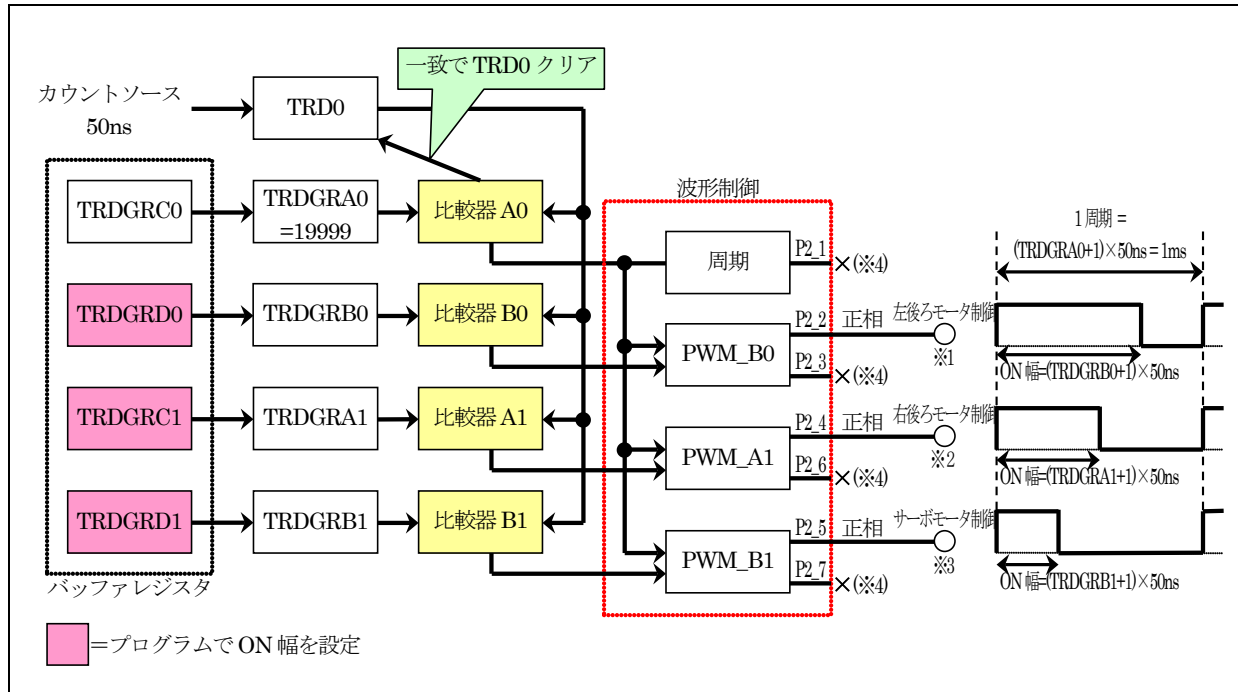
380 :      /* タイマRD リセット同期PWMモード設定(左後モータ、右後モータ、サーボモータ) */
381 :      trdpsr0 = 0x08;                          /* TRDIOB0, C0, D0端子設定 */
382 :      trdpsr1 = 0x05;                          /* TRDIOA1, B1, C1, D1端子設定 */
383 :      trdmr  = 0xf0;                          /* バッファレジスタ設定 */
384 :      trdfcr = 0x01;                          /* リセット同期PWMモードに設定 */
385 :      trdcr0 = 0x20;                          /* ソースカウントの選択:f1 */
386 :      trdgra0 = trdgrc0 = TRD_MOTOR_CYCLE - 1; /* 周期設定 */
387 :      trdgrb0 = trdgrd0 = 0;                  /* P2_2端子のON幅(左後モータ) */
388 :      trdgral = trdgrcl = 0;                  /* P2_4端子のON幅(右後モータ) */
389 :      trdgrbl = trdgrdl = 0;                  /* P2_5端子のON幅(サーボモータ) */
390 :      trdoerl = 0xcd;                          /* 出力端子の選択 */
391 :      trdstr  = 0x0d;                          /* TRD0カウント開始 */
    
```

タイマRDのリセット同期PWMモードを使うと、同一周期のPWM波形を3本出力できます。ただし、周期やPWMのON幅を設定するタイミングによっては、波形が乱れる場合がありますので、バッファレジスタを使って対処します。

タイマRDのリセット同期PWMモードを使ったPWM波形出力の代表的な様子を、下図に示します。



今回の設定をした PWM 波形が出力される様子を、下図に示します。



※1…この端子は、TRDPSR0 で端子の割り当て、TRDFCR で初期出力と出力レベルの設定、TRDOER1 で出力許可することによって、PWM 波形が出力されます。

※2…この端子は、TRDPSR1 で端子の割り当て、TRDFCR で初期出力と出力レベルの設定、TRDOER1 で出力許可することによって、PWM 波形が出力されます。

※3…この端子は、TRDPSR1 で端子の割り当て、TRDFCR で初期出力と出力レベルの設定、TRDOER1 で出力許可することによって、PWM 波形が出力されます。

※4…この端子は、PWM 波形出力する設定にはしません。通常の I/O ポートとして使用できます。

6. マイコンカー走行プログラムの解説

(1) タイマ RD 端子選択レジスタ 0 (TRDPSR0: Timer RD function select register 0) の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x08
6	trdiiod0sel0	TRDIOD0 端子選択ビットを設定します。 0: TRDIOD0 端子は使用しない 1: P2_3 に割り当てる	
5	trdioc0sel1	TRDIOC0 端子選択ビットを設定します。 00: TRDIOC0 端子は使用しない 01: 設定しないでください	
4	trdioc0sel0	10: P2_1 に割り当てる 11: 設定しないでください	
3	trdiob0sel1	TRDIOB0 端子選択ビットを設定します。 00: TRDIOB0 端子は使用しない 01: 設定しないでください	
2	trdiob0sel0	10: P2_2 に割り当てる 11: 設定しないでください	
1	-	"0"を設定	
0	trdioa0sel0	TRDIOA0/TRDCLK 端子選択ビットを設定します。 0: TRDIOA0/TRDCLK 端子は使用しない 1: P2_0 に割り当てる	

(2) タイマ RD 端子選択レジスタ 1 (TRDPSR1: Timer RD function select register 1) の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x05
6	trdiiod1sel0	TRDIOD1 端子選択ビットを設定します。 0: TRDIOD1 端子は使用しない 1: P2_7 に割り当てる	
5	-	"0"を設定	
4	trdioc1sel0	TRDIOC1 端子選択ビットを設定します。 0: TRDIOC1 端子は使用しない 1: P2_6 に割り当てる	
3	-	"0"を設定	
2	trdiob1sel0	TRDIOB1 端子選択ビットを設定します。 0: TRDIOB1 端子は使用しない 1: P2_5 に割り当てる	
1	-	"0"を設定	
0	trdioa1sel0	TRDIOA1 端子選択ビットを設定します。 0: TRDIOA1 端子は使用しない 1: P2_4 に割り当てる	

(3) タイマ RD モードレジスタ(TRDMDR: Timer RD mode register)の設定

ビット	シンボル	説明	設定値
7	bfd1_trdmr	TRDGRD1 レジスタ機能選択ビットを設定します。 0: ジェネラルレジスタ 1: TRDGRB1 レジスタのバッファレジスタ	0xf0
6	bfc1_trdmr	TRDGRC1 レジスタ機能選択ビットを設定します。 0: ジェネラルレジスタ 1: TRDGRA1 レジスタのバッファレジスタ	
5	bfd0_trdmr	TRDGRD0 レジスタ機能選択ビットを設定します。 0: ジェネラルレジスタ 1: TRDGRB0 レジスタのバッファレジスタ	
4	bfc0_trdmr	TRDGRC0 レジスタ機能選択ビットを設定します。 0: ジェネラルレジスタ 1: TRDGRA0 レジスタのバッファレジスタ	
3	-	"0"を設定	
2	-	"0"を設定	
1	-	"0"を設定	
0	-	"0"を設定	

(4) タイマ RD 機能制御レジスタ(TRDFCR: Timer RD function control register)の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x01
6	stclk_trdfer	外部クロック入力選択ビットを設定します。 0: 外部クロック入力無効 1: 外部クロック入力有効	
5	-	"0"を設定	
4	-	"0"を設定	
3	ols1_trdfer	逆相出力レベル選択ビットを設定します。 0: 初期出力“H”、アクティブレベル“L” 1: 初期出力“L”、アクティブレベル“H”	
2	ols0_trdfer	正相出力レベル選択ビットを設定します。 0: 初期出力“H”、アクティブレベル“L” 1: 初期出力“L”、アクティブレベル“H”	
1	cmd1_trdfer	コンビネーションモード選択ビットを設定します。	
0	cmd0_trdfer	リセット同期 PWM モードでは“01”(リセット同期 PWM モード)にしてください。	

6. マイコンカー走行プログラムの解説

(5) タイマ RD 制御レジスタ 0(TRDCR0: Timer RD control register 0)の設定

ビット	シンボル	説明	設定値
7	cclr2_trdcr0	TRD0 カウンタクリア選択ビットを設定します。	0x20
6	cclr1_trdcr0	リセット同期 PWM モードの場合は、“001”(TRDGRA0 とのコンペア一致で TRD0 レジスタクリア)に設定します。	
5	cclr0_trdcr0		
4	ckeg1_trdcr0		
3	ckeg0_trdcr0	10: 両エッジでカウント 11: 設定しないでください	
2	tck2_trdcr0	カウントソース選択ビットを設定します。 000: f1 (1/20MHz=50ns) 001: f2 (2/20MHz=100ns) 010: f4 (4/20MHz=200ns) 011: f8 (8/20MHz=400ns) 100: f32 (32/20MHz=1600ns)	
1	tck1_trdcr0	101: TRDCLK 入力または fC2(2/XCIN クロック=今回は未接続) 110: fOCO40M (高速オンチップオシレータ 40MHz=今回は未接続) 111: fOCO-F (高速オンチップオシレータを FRA2 で分周したクロック=今回は未接続)	
0	tck0_trdcr0		

(6) タイマ RD ジェネラルレジスタ A0(TRDGRA0: Timer RD General register A0)、タイマ RD ジェネラルレジスタ C0(TRDGRC0: Timer RD General register C0)の設定

ビット	シンボル	説明	設定値
15~0	-	<p>TRDGRA0 は、PWM 波形の周期を設定します。計算式を下記に示します。</p> <p>$TRDGRA0 = \text{PWM 波形の周期} / \text{タイマ RD カウンタのカウントソース} - 1$</p> <p>今回、タイマ RD の PWM 波形の周期は 1ms に設定します。タイマ RD カウンタのカウントソースは、TRDCR0 で設定した 50ns です。よって、</p> $TRDGRA0 = (1 \times 10^{-3}) / (50 \times 10^{-9}) - 1$ $= 20000 - 1$ <p>となります。</p> <p>プログラムでは、define 文を使って</p> <pre>30 : #define TRD_MOTOR_CYCLE 20000 /* 左後,右後,サ-ボモータ PWM の周期 */ とタイマ RD の PWM 波形の周期を記号定数で設定しています。よってプログラムでは、</pre> <pre>386 : trdgra0 = trdgrc0 = TRD_MOTOR_CYCLE - 1; /* 周期設定 */</pre> <p>としています。</p> <p>TRDGRC0 は、TRDGRA0 のバッファレジスタで、次の役割があります。</p> <p>① TRDGRA0 の値を変えるとき、設定するタイミングによっては、PWM 波形が乱れることがあります。</p> <p>② それを防止するために、TRDGRC0 に値を設定します。TRDGRA0 の設定は今回だけで、これ以降はこのレジスタの値は変更しません。</p> <p>③ 「TRDGRA0+1=TRD0」になった瞬間、TRDGRA0 に TRDGRC0 の値がコピーされます。</p>	19999

(7) タイマ RD ジェネラルレジスタ B0(TRDGRB0:Timer RD General register B0)、タイマ RD ジェネラルレジスタ D0(TRDGRD0:Timer RD General register D0)の設定

ビット	シンボル	説明	設定値
15~0	-	<p>TRDGRB0 は、P2_2 端子の ON 幅を設定します。今回は、左後ろモータの PWM 波形の ON 幅を設定します。P2_3 端子からは、その反転した波形が出力されますが、今回は波形出力を禁止にして、通常の I/O ポートとして使用します。計算式を下記に示します。</p> <p>$TRDGRB0 = P2_2 \text{ 端子の PWM 波形の ON 幅} / \text{タイマ RD カウンタのカウントソース} - 1$</p> <p>ただし、設定によっては下記のようになります。</p> <p>①TRDGRB0 の値を、TRDGRA0 と同じ値にすると ON 幅は 0%になる ②TRDGRB0 の値を、TRDGRA0+1 より大きい値にすると ON 幅は 100%になる</p> <p>ここではまだ ON 幅は 0%にするので、0 を設定します。</p> <p>TRDGRD0 は、TRDGRB0 のバッファレジスタで、次の役割があります。</p> <p>①TRDGRB0 の値を変えるとき、設定するタイミングによっては、ON 幅 100%の波形が 1 周期出力されてしまうことがあります。 ②それを防止するために、TRDGRD0 に値を設定します。TRDGRB0 の設定は今回だけで、これ以降はこのレジスタの値は変更しません。 ③「TRDGRA0+1=TRD0」になった瞬間、TRDGRB0 に TRDGRD0 の値がコピーされます。</p>	0

(8) タイマ RD ジェネラルレジスタ A1(TRDGRA1:Timer RD General register A1)、タイマ RD ジェネラルレジスタ C1(TRDGRC1:Timer RD General register C1)の設定

ビット	シンボル	説明	設定値
15~0	-	<p>TRDGRA1 は、P2_4 端子の ON 幅を設定します。今回は、右後ろモータの PWM 波形の ON 幅を設定します。P2_6 端子からは、その反転した波形が出力されますが、今回は波形出力を禁止にして、通常の I/O ポートとして使用します。計算式を下記に示します。</p> <p>$TRDGRA1 = P2_4 \text{ 端子の PWM 波形の ON 幅} / \text{タイマ RD カウンタのカウントソース} - 1$</p> <p>TRDGRC1 は、TRDGRA1 のバッファレジスタです。 TRDGRA1、TRDGRC1 の設定、役割は TRDGRB0、TRDGRD0 と同じです。</p>	0

6. マイコンカー走行プログラムの解説

(9) タイマ RD ジェネラルレジスタ B1(TRDGRB1:Timer RD General register B1)、タイマ RD ジェネラルレジスタ D1(TRDGRD1:Timer RD General register D1)の設定

ビット	シンボル	説明	設定値
15~0	-	<p>TRDGRB1 は、P2.5 端子の ON 幅を設定します。今回は、自作サーボモータの PWM 波形の ON 幅を設定します。P2.7 端子からは、その反転した波形が出力されますが、今回は波形出力を禁止にして、通常の I/O ポートとして使用します。計算式を下記に示します。</p> <p>TRDGRA1=P2_4 端子の PWM 波形の ON 幅/タイマ RD カウンタのカウントソース-1</p> <p>TRDGRD1 は、TRDGRB1 のバッファレジスタです。 TRDGRB1、TRDGRD1 の設定、役割は TRDGRB0、TRDGRD0 と同じです。</p>	0

(10) タイマ RD アウトプットマスタ許可レジスタ 1(TRDOER1:Timer RD output master enable register 1)の設定

ビット	シンボル	説明	設定値
7	ed1_trdoer1	<p>TRDIOD1(P2_7)出力禁止ビットを設定します。 0:出力許可 1:出力禁止(TRDIOD1 端子はプログラマブル入出力ポート)</p>	0xcd
6	ec1_trdoer1	<p>TRDIOC1(P2_6)出力禁止ビットを設定します。 0:出力許可 1:出力禁止(TRDIOC1 端子はプログラマブル入出力ポート)</p>	
5	eb1_trdoer1	<p>TRDIOB1(P2_5)出力禁止ビットを設定します。 0:出力許可 1:出力禁止(TRDIOB1 端子はプログラマブル入出力ポート)</p>	
4	ea1_trdoer1	<p>TRDIOA1(P2_4)出力禁止ビットを設定します。 0:出力許可 1:出力禁止(TRDIOA1 端子はプログラマブル入出力ポート)</p>	
3	ed0_trdoer1	<p>TRDIOD0(P2_3)出力禁止ビットを設定します。 0:出力許可 1:出力禁止(TRDIOD0 端子はプログラマブル入出力ポート)</p>	
2	ec0_trdoer1	<p>TRDIOC0(P2_1)出力禁止ビットを設定します。 0:出力許可 1:出力禁止(TRDIOC0 端子はプログラマブル入出力ポート)</p>	
1	eb0_trdoer1	<p>TRDIOB0(P2_2)出力禁止ビットを設定します。 0:出力許可 1:出力禁止(TRDIOB0 端子はプログラマブル入出力ポート)</p>	
0	-	"1"を設定	

(11) タイマ RD スタートレジスタ (TRDSTR: Timer RD start register) の設定

ビット	シンボル	説明	設定値
7	-	"0"を設定	0x0d
6	-	"0"を設定	
5	-	"0"を設定	
4	-	"0"を設定	
3	cse11_trdstr	TRD1 カウント動作選択ビットを設定します。 0: TRDGRA1 レジスタとのコンペアー一致でカウント停止 1: TRDGRA1 レジスタとのコンペアー一致後もカウント継続	
2	cse10_trdstr	TRD0 カウント動作選択ビットを設定します。 0: TRDGRA0 レジスタとのコンペアー一致でカウント停止 1: TRDGRA0 レジスタとのコンペアー一致後もカウント継続	
1	tstart1_trdstr	TRD1 カウント開始フラグを設定します。 0: カウント停止 1: カウント開始	
0	tstart0_trdstr	TRD0 カウント開始フラグを設定します。 0: カウント停止 1: カウント開始	

(12) 出力されるタイミングと初期出力、アクティブレベルについて

※正相出力レベル選択ビットと出力波形の関係

タイマ RD 機能制御レジスタ (TRDFCR) の正相出力レベル選択ビットと出力波形の関係を、下図に示します。
正相出力とは、P2_2 端子、P2_4 端子、P2_5 端子から出力される波形のことです。

bit2	波形	説明
0 (今回)		"1" からスタートする設定です。
1		"0" からスタートする設定です。

6. マイコンカー走行プログラムの解説

※逆相出力レベル選択ビットと出力波形の関係

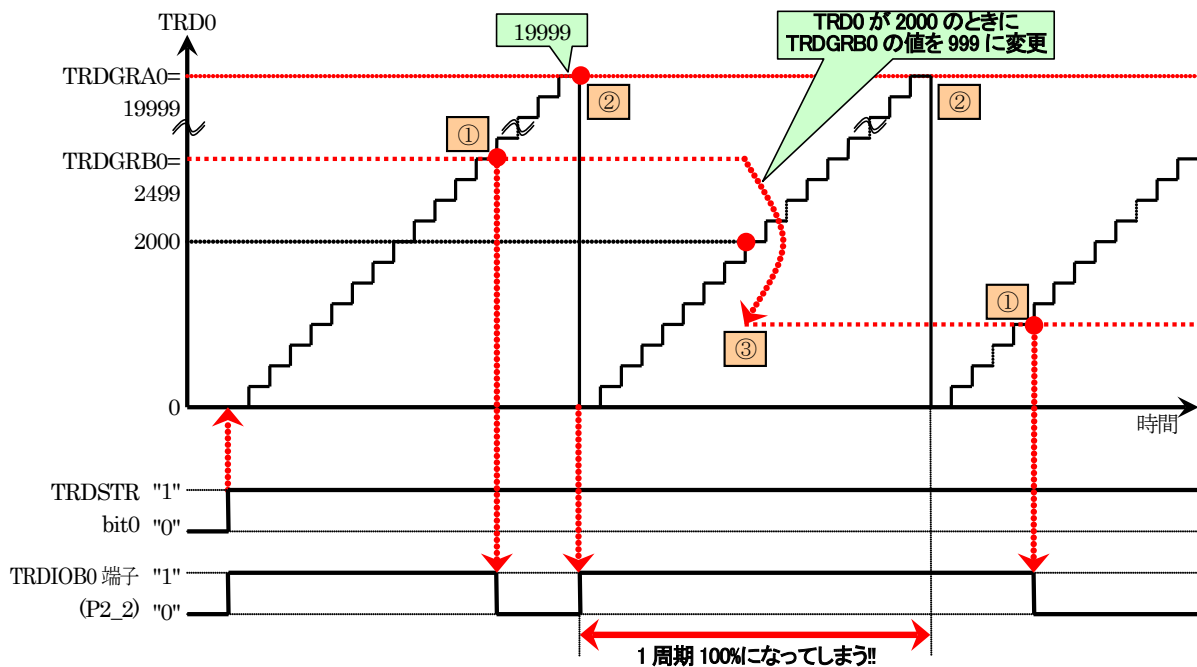
タイマ RD 機能制御レジスタ (TRDFCR) の逆相出力レベル選択ビットと出力波形の関係を、下図に示します。

逆相出力とは、P2_3 端子、P2_6 端子、P2_7 端子から出力される波形のことです。今回は、逆相出力波形は出力していません。

bit3	波形	説明
0 (標準)	<p>出力波形</p> <p>TRDSTR の bit0="1" (PWM 出力開始)</p>	"0" からスタートする設定です。
1	<p>出力波形</p> <p>TRDSTR の bit0="1" (PWM 出力開始)</p>	"1" からスタートする設定です。

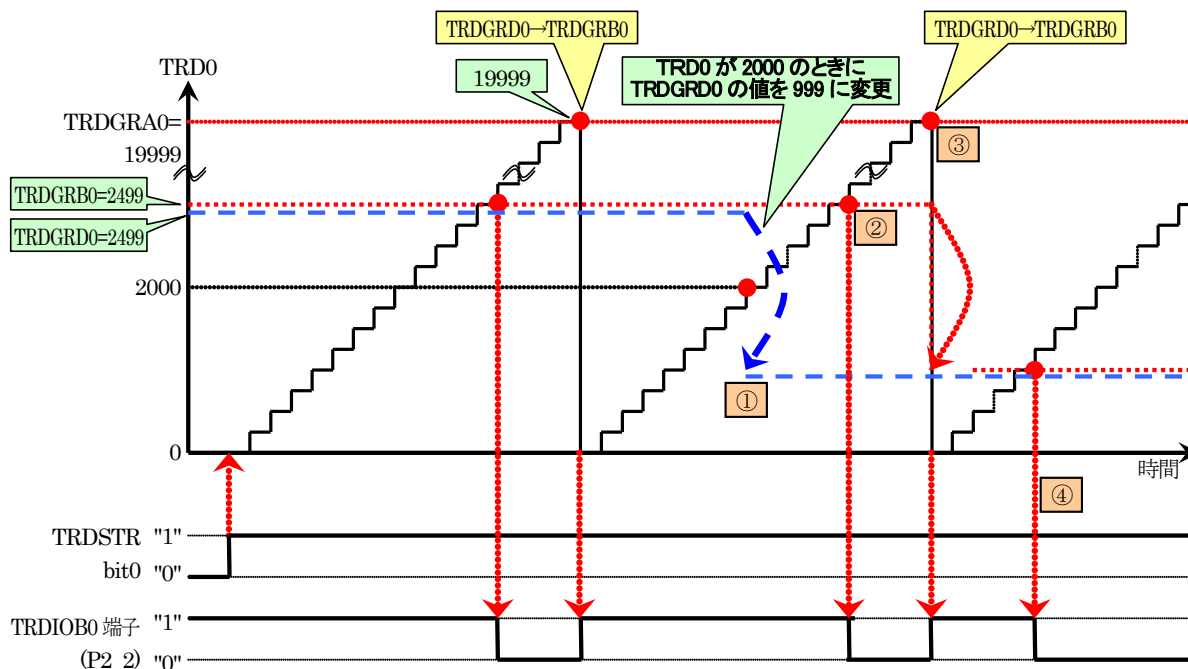
(13) PWM 波形

各レジスタの値と P2_2 端子の PWM 波形の様子を、下図に示します。



①	①TRD0=TRDGRB0+1 になった瞬間、“0”になります。
②	②TRD0=TRDGRA0+1 になった瞬間、“1”になります。TRD0 はこのタイミングで 0 になります。PWM 波形は、下記の①と②を繰り返すことにより、PWM 波形が出力されます。
③	ON 幅を変えるときは、プログラムで TRDGRB0 の値を変えます。このとき、図の③のタイミングで TRDGRB0 の値を変えてしまった場合、①の波形が“0”になるタイミングがなく、1 周期 100%の波形が出力されてしまいます。

そこで、TRDGRB0 の値は直接変更せず、バッファレジスタを使います。バッファレジスタへの設定を含めた P2_2 端子の PWM 波形の様子を、下図に示します。



①	プログラムでは TRDGRB0 の値ではなく、TRDGRD0 の値を変えます。
②	TRDGRB0 の値は変わっていないため「TRD0=TRDGRB0+1」になり、波形は「0」になります。
③	<ul style="list-style-type: none"> ・「TRD0=TRDGRA0+1」になり、波形は「1」になります。 ・TRD0 の値が 0 になります。 ・TRDGRD0 の値が、TRDGRB0 へ転送されます。このタイミングは、1 周期 100%の波形が出力されないタイミングです。
④	「TRD0=TRDGRB0+1」になり、波形は「0」になります。

このようにバッファレジスタを使うことにより 1 周期 100%の波形が出力される問題を解決します。周期、ON 幅を決めるレジスタと、バッファレジスタの関係を下表に示します。

周期、ON 幅を決めるレジスタ名	バッファレジスタ
TRDGRA0	TRDGRC0
TRDGRB0	TRDGRD0
TRDGRA1	TRDGRC1
TRDGRB1	TRDGRD1

周期、ON 幅を決めるレジスタは、init 関数などいちばん最初だけ設定、2 回目以降はバッファレジスタに値を設定します。

6. マイコンカー走行プログラムの解説

6.2.11 タイマ RB の 1ms ごとの割り込みプログラム

```

394 : /*****/
395 : /* タイマRB 割り込み処理 */
396 : /*****/
397 : #pragma interrupt /B intTRB(vect=24)
398 : void intTRB( void )
399 : {
400 :     unsigned int i;
401 :
402 :     asm(" fset I ");          /* タイマRB以上の割り込み許可 */
403 :
404 :     cnt1++;
405 :
406 :     /* サーボモータ制御 */
407 :     servoControl();
408 :
409 :     /* ブザー処理 */
410 :     beepProcessS();

```

402 行	タイマ RB 割り込み以上のレベルの割り込みがあったときに、割り込み処理に移れるよう割り込みを許可しておきます。割り込みプログラムが実行された時点で割り込みが禁止されるので、割り込みプログラム内で割り込みを許可する場合は、改めて割り込みを許可しておかなければ行けません。
404 行	cnt1 変数の値を 1 つ増やします。よって、cnt1 は 1ms ごとに増加していきます。
407 行	サーボモータの PWM 値を計算します。
410 行	ブザーを鳴らす処理を行います。

```

412 :     /* 10回中1回実行する処理 */
413 :     iTimer10++;
414 :     switch( iTimer10 ) {
415 :     case 1:
416 :         /* エンコーダ制御 */
417 :         i = trg;
418 :         iEncoder      = i - uEncoderBuff;
419 :         lEncoderTotal += iEncoder;
420 :         uEncoderBuff = i;
421 :         break;

```

中略

```

455 :     case 10:
456 :         /* iTimer10変数の処理 */
457 :         iTimer10 = 0;
458 :         break;
459 :     }
460 : }

```

413 行	iTimer10 変数の値を 1 つ増やします。
414~ 459 行	iTimer10 変数の値に応じて分岐させます。case 文は 1~10 まであり iTimer10 変数は 1ms ごとに増えていくので、10ms ごとに 1 回 case 文が順番に実行されることになります。

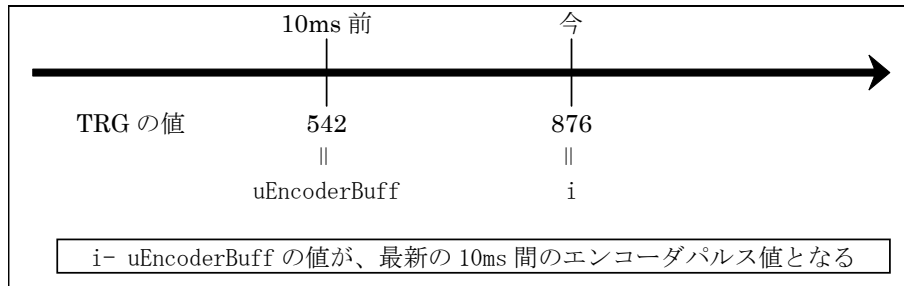
ロータリエンコーダに関する変数処理です。
 タイマ RG カウンタ(TRG)の値が、ロータリエンコーダの積算パルス数です。範囲は 0~65535 で、65535 の次は 0 になります。

iEncoder..... 10ms 間のロータリエンコーダのパルス数を入力します。
 lEncoderTotal... ロータリエンコーダの積算値を入力します。long 型なので約 21 億パルスまでカウントできます。

iEncoder 値は、次の計算で求めます。

$$iEncoder = \text{現在の TRG の値}(i \text{ 変数}) - 10\text{ms 前の TRG の値}(uEncoderBuff \text{ 変数})$$

計算の様子を、下図に示します。



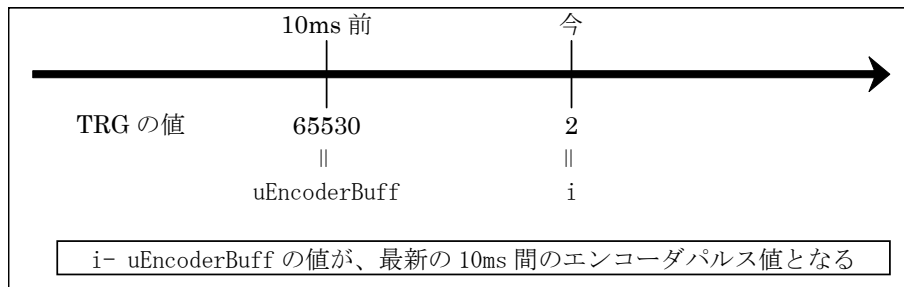
lEncoderTotal 変数は、10ms ごとに iEncoder の値を加えていきます。
 416 行で、現在の TRG の値(i 変数)を、uEncoderBuff 変数に代入します。10ms 後は、uEncoderBuff 変数の値が、10ms 前の値になっています。
 今回、i 変数に TRG の値を一度コピーしてから 10ms 前の値の差分を計算して、iEncoder に代入しています。下記のプログラムのように、TRG の値を直接使った方が分かりやすいですが、そうしていません。

416~
421 行

```
1 : iEncoder = trg;
2 : trg = 0;
3 : lEncoderTotal += iEncoder;
```

これは、1 行目が終わってから 2 行目の TRG の値をクリアするまでの間に、パルスがカウントされてしまった場合、カウント洩れが起きてしまうためです。それを防ぐためにちょっと複雑ですが、今回のようなプログラムにしています。

TRG の上限である、65535 を超えた場合はどうなるのでしょうか。



$$iEncoder = \text{現在の TRG の値}(i \text{ 変数}) - 10\text{ms 前の TRG の値}(uEncoderBuff \text{ 変数})$$

$$iEncoder = 2 - 65530 = -62228$$

この値を、16 進数で表すと 0xffff0008 となります。変数の型は、符号無し 16bit 幅なので、下位の 16bit のみ有効となり 0x0008=8 となります。カウント回数としては、「65531、65532、65533、65534、65535、0、1、2」の 8 カウント分となり、計算結果と一致します。

6. マイコンカー走行プログラムの解説

```

423 :     case 2:
424 :         /* スイッチ読み込み準備 */
425 :         p9_4 = 0;                               /* LED出力OFF          */
426 :         pd8 = 0x00;
427 :         break;
    
```

425～
427 行 ポート 8 に接続されているディップスイッチの値を読み込む準備です。
 P9_4 を"0"にしてポート 8 に接続されている LED を消灯します。
 PD8 を 0x00 にして、ポート 8 を入力設定にします。

```

429 :     case 3:
430 :         /* スイッチ読み込み、LED出力 */
431 :         types_dipsw = ~p8;                       /* ドライブ基板TypeS Ver.3のSW読み込み*/
432 :         p8 = types_led;                           /* ドライブ基板TypeS Ver.3のLEDへ出力*/
433 :         pd8 = 0xff;
434 :         p9_4 = 1;                               /* LED出力ON          */
435 :         break;
    
```

431～
435 行 ポート 8 に接続されているディップスイッチの値を types_dipsw 変数に代入します。
 ポート 8 に LED 出力したい値(types_led 変数の値)を代入します。
 PD8 を 0xff にして、ポート 8 を出力設定にします。
 P9_4 を"1"にしてポート 8 に接続されている LED を点灯します。
 このように、iTimer10 変数が 2 のときは LED が消灯、2 以外のときは LED を点灯させます。

```

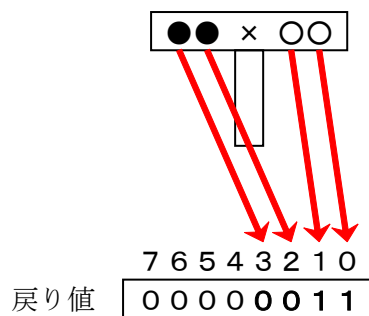
437 :     case 4:
438 :         break;
439 :
440 :     case 5:
441 :         break;
442 :
443 :     case 6:
444 :         break;
445 :
446 :     case 7:
447 :         break;
448 :
449 :     case 8:
450 :         break;
451 :
452 :     case 9:
453 :         break;
454 :
455 :     case 10:
456 :         /* iTimer10変数の処理 */
457 :         iTimer10 = 0;
458 :         break;
459 :     }
460 : }
    
```

437～
459 行 4～9 は何もしていません。今後、10ms ごとに処理したい内容があれば、この中にプログラムを追加
 してください。457 行は iTimer10 変数を 0 にして case 文が再度実行されるようにします。

6.2.12 アナログセンサ基板 TypeS Ver.2 のデジタルセンサ値読み込み

```
475 : /*****  
476 : /* アナログセンサ基板TypeS Ver.2のデジタルセンサ値読み込み */  
477 : /* 引数 なし */  
478 : /* 戻り値 左端、左中、右中、右端のデジタルセンサ 0:黒 1:白 */  
479 : *****/  
480 : unsigned char sensor_inp( void )  
481 : {  
482 :     unsigned char sensor;  
483 :  
484 :     sensor = ~p0 & 0x0f;  
485 :  
486 :     return sensor;  
487 : }
```

アナログセンサ基板 TypeS Ver.2 のデジタルセンサ 4 個を読み込む関数です。デジタルセンサは黒で“1”、白で“0”なのでポートから読み込むときに“~”(チルダ)をつけて反転させます。



※中心のデジタルセンサ値を読み込む関数は、center_inp 関数です。

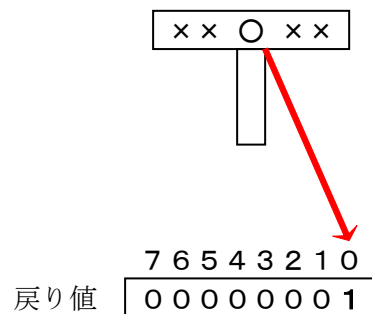
6. マイコンカー走行プログラムの解説

6.2.13 アナログセンサ基板 TypeS Ver.2 の中心デジタルセンサ読み込み

```

489 : /*****/
490 : /* アナログセンサ基板TypeS Ver.2の中心デジタルセンサ読み込み */
491 : /* 引数 なし */
492 : /* 戻り値 中心デジタルセンサ 0:黒 1:白 */
493 : /*****/
494 : unsigned char center_inp( void )
495 : {
496 :     unsigned char sensor;
497 :
498 :     sensor = ~p1_7 & 0x01;
499 :
500 :     return sensor;
501 : }
    
```

アナログセンサ基板 TypeS Ver.2 の中心デジタルセンサ 1 個を読み込む関数です。



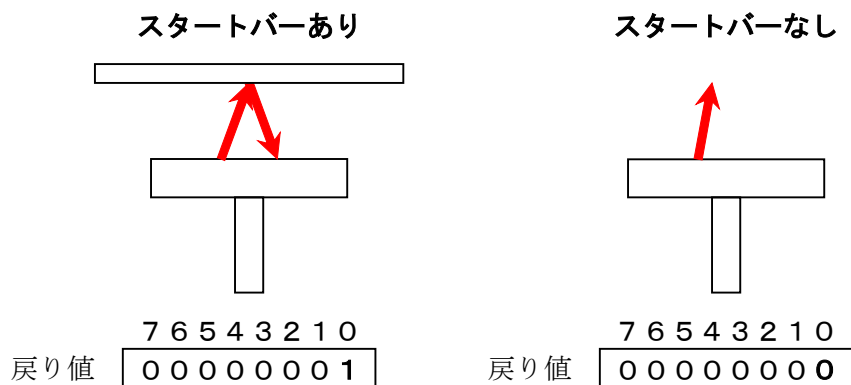
※中心以外のデジタルセンサ値を読み込む関数は、sensor_inp 関数です。

6.2.14 アナログセンサ基板 TypeS Ver.2 のスタートバー検出センサ読み込み

```

503 : /*****/
504 : /* アナログセンサ基板TypeS Ver.2のスタートバー検出センサ読み込み */
505 : /* 引数 なし */
506 : /* 戻り値 0:スタートバーなし 1:スタートバーあり */
507 : /*****/
508 : unsigned char startbar_get( void )
509 : {
510 :     unsigned char sensor;
511 :
512 :     sensor = ~p1_6 & 0x01;
513 :
514 :     return sensor;
515 : }
    
```

アナログセンサ基板 TypeS Ver.2 のスタートバー検出センサの状態を読み込む関数です。



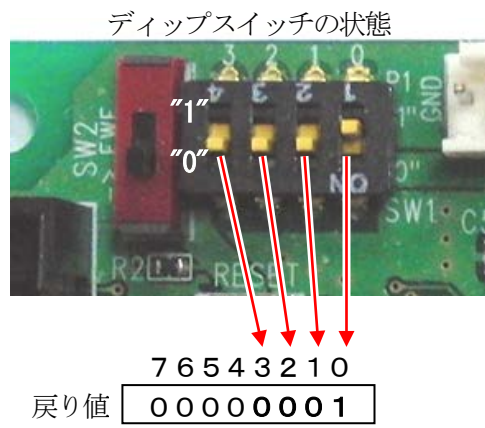
6. マイコンカー走行プログラムの解説

6.2.15 RY_R8C38 ボード上のディップスイッチ値読み込み

```

517 : /*****
518 : /* マイコンボード上のディップスイッチ値読み込み */
519 : /* 引数 なし */
520 : /* 戻り値 スイッチ値 0~15 */
521 : *****/
522 : unsigned char dipsw_get( void )
523 : {
524 :     unsigned char sw;
525 :
526 :     sw = p1 & 0x0f;          /* P1_3~P1_0読み込み */
527 :
528 :     return sw;
529 : }
    
```

RY_R8C38 ボードのディップスイッチの値を読み込む関数です。戻り値は、0~15 (2^4-1)です。

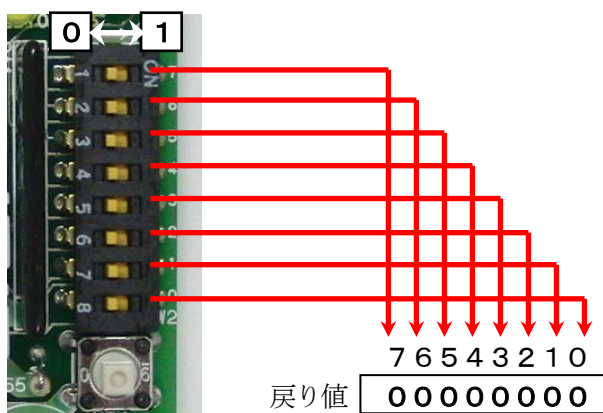


6.2.16 モータドライブ基板 TypeS Ver.3 上のディップスイッチ値読み込み

```

531 : /*****/
532 : /* モータドライブ基板TypeS Ver. 3上のディップスイッチ値読み込み */
533 : /* 引数 なし */
534 : /* 戻り値 スイッチ値 0~255 */
535 : /*****/
536 : unsigned char dipsw_get2( void )
537 : {
538 :     /* 実際の入力はタイマRB割り込み処理で実施 */
539 :     return types_dipsw;
540 : }
    
```

モータドライブ基板 TypeS Ver.3 上のディップスイッチの値を読み込む関数です。戻り値は、0~255 (2⁸-1)です。



6. マイコンカー走行プログラムの解説

6.2.17 モータドライブ基板 TypeS Ver.3 上のプッシュスイッチ値読み込み

```

542 : /*****/
543 : /* モータドライブ基板TypeS Ver.3上のプッシュスイッチ値読み込み */
544 : /* 引数 なし */
545 : /* 戻り値 スイッチ値 0:OFF 1:ON */
546 : /*****/
547 : unsigned char pushsw_get( void )
548 : {
549 :     unsigned char sw;
550 :
551 :     sw = ~p9_5 & 0x01;
552 :
553 :     return sw;
554 : }
    
```

モータドライブ基板 TypeS Ver.3 上のプッシュスイッチの値を読み込む関数です。

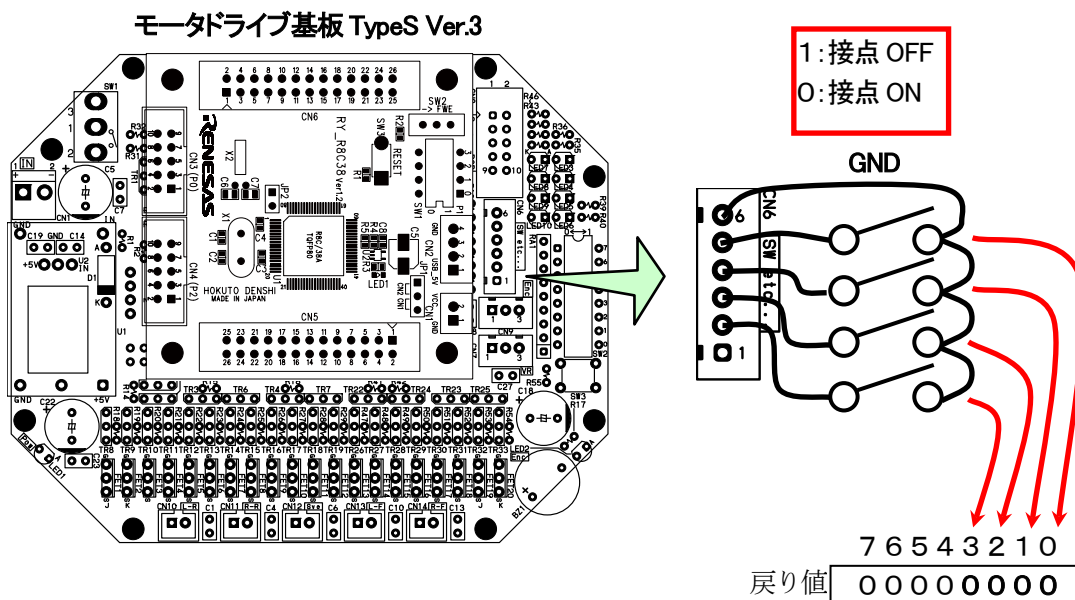


6.2.18 モータドライブ基板 TypeS Ver.3 の CN6 の状態読み込み

```

556 : /*****
557 : /* モータドライブ基板TypeS Ver. 3のCN6の状態読み込み */
558 : /* 引数 なし */
559 : /* 戻り値 0~15 */
560 : *****/
561 : unsigned char cn6_get( void )
562 : {
563 :     unsigned char data;
564 :
565 :     data = p7 >> 4;
566 :
567 :     return data;
568 : }
    
```

モータドライブ基板 TypeS Ver.3 の CN6 の状態を読み込む関数です。



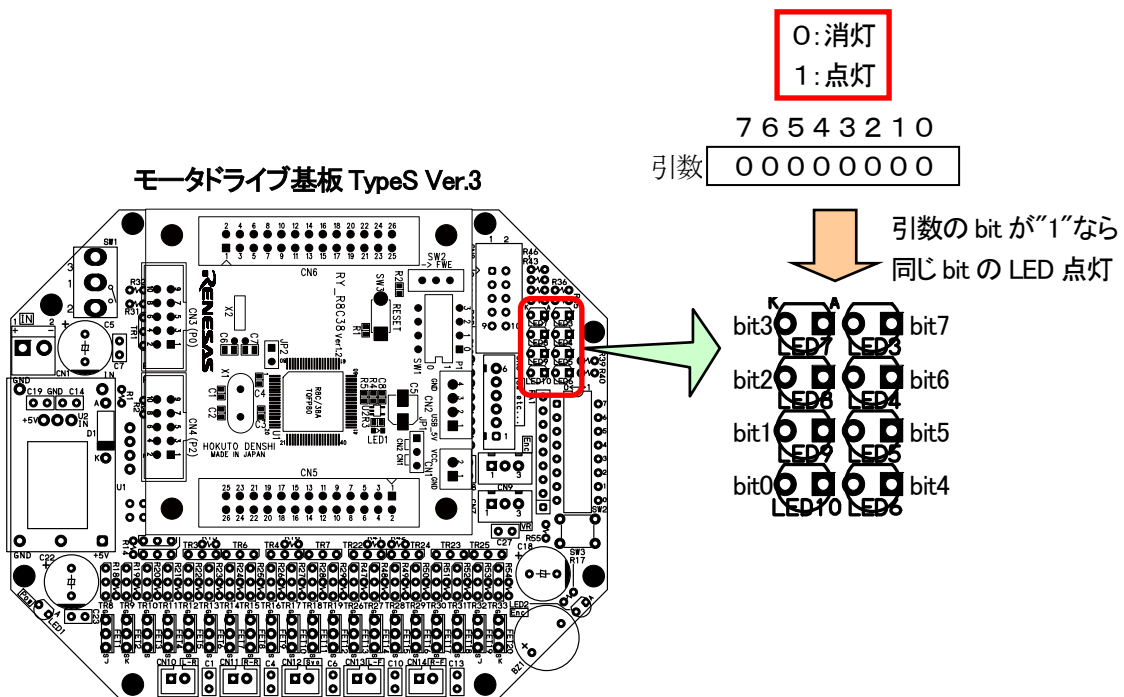
6. マイコンカー走行プログラムの解説

6.2.19 モータドライブ基板 TypeS Ver.3 の LED 制御

```

570 : /*****/
571 : /* モータドライブ基板TypeS Ver. 3のLED制御 */
572 : /* 引数 8個のLED制御 0:OFF 1:ON */
573 : /* 戻り値 なし */
574 : /*****/
575 : void led_out( unsigned char led )
576 : {
577 :     /* 実際の出力はタイマRB割り込み処理で実施 */
578 :     types_led = led;
579 : }
    
```

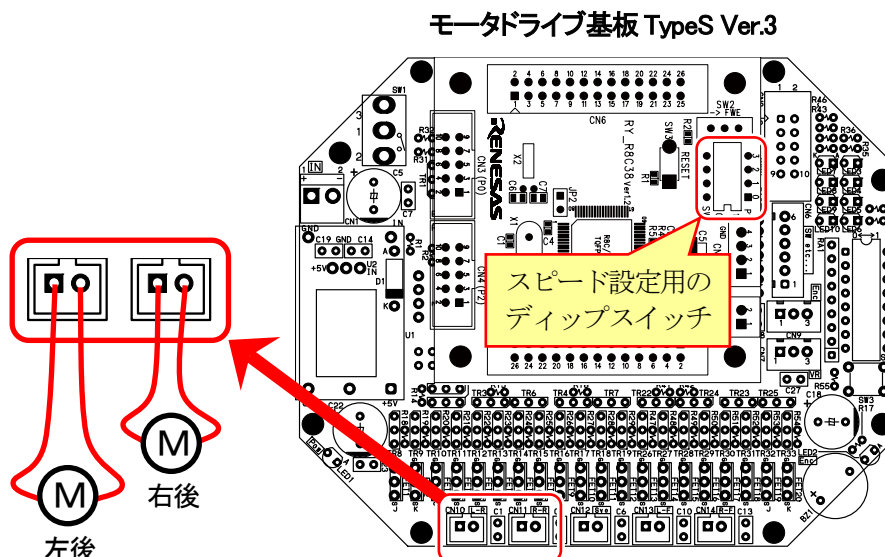
モータドライブ基板 TypeS Ver.3 の 8 個の LED を制御する関数です。



6.2.20 後輪の速度制御

```
581 : /*****  
582 : /* 後輪の速度制御 */  
583 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */  
584 : /*      0で停止、100で正転100%、-100で逆転100% */  
585 : /* 戻り値 なし */  
586 : /*****  
587 : void motor_r( int accele_l, int accele_r )  
588 : {  
589 :     int sw_data;  
590 :  
591 :     sw_data = dipsw_get() + 5;          /* ディップスイッチ読み込み */  
592 :     accele_l = accele_l * sw_data / 20;  
593 :     accele_r = accele_r * sw_data / 20;  
594 :  
595 :     /* 左後モータ */  
596 :     if( accele_l >= 0 ) {  
597 :         p2_1 = 0;  
598 :         trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_l / 100;  
599 :     } else {  
600 :         p2_1 = 1;  
601 :         trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_l ) / 100;  
602 :     }  
603 :  
604 :     /* 右後モータ */  
605 :     if( accele_r >= 0 ) {  
606 :         p2_3 = 0;  
607 :         trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_r / 100;  
608 :     } else {  
609 :         p2_3 = 1;  
610 :         trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_r ) / 100;  
611 :     }  
612 : }
```

モータドライブ基板 TypeS Ver.3 の後輪モータ 2 個を制御する関数です。



使い方を下記に示します。

```
motor_r ( 左後モータの PWM, 右後モータの PWM );
```

左後モータの PWM、右後モータの PWM の値は、下記を設定することができます。

- 0… 停止
- 1～100… 正転の割合 100 が一番速い
- 1～-100… 逆転の割合 100 が一番速い

モータへの出力は、motor_r 関数で設定した割合がそのままモータに出力されるのではなく、下記の計算結果がモータに出力されます。

実際のモータに出力される割合 = 引数 × (マイコンボードのディップスイッチの値 + 5) ÷ 20

例えば、ディップスイッチの値が 10 で下記プログラムを実行したとします。

```
motor_r( 50, 100 );
```

左後モータに出力される割合 = プログラムの割合 × (ディップスイッチの値 + 5) ÷ 20
 = 50 × (10 + 5) ÷ 20
 = 37.5
 ≒ **37** (小数点以下は切り捨てです)

右後モータに出力される割合 = プログラムの割合 × (ディップスイッチの値 + 5) ÷ 20
 = 100 × (10 + 5) ÷ 20
 = **75**

6.2.21 後輪の速度制御 2 ディップスイッチには関係しない motor 関数

```
614 : /*****  
615 : /* 後輪の速度制御2 ディップスイッチには関係しないmotor関数 */  
616 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */  
617 : /*      0で停止、100で正転100%、-100で逆転100% */  
618 : /* 戻り値 なし */  
619 : /*****  
620 : void motor2_r( int accele_l, int accele_r )  
621 : {  
622 :     /* 左後モータ */  
623 :     if( accele_l >= 0 ) {  
624 :         p2_1 = 0;  
625 :         trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_l / 100;  
626 :     } else {  
627 :         p2_1 = 1;  
628 :         trdgrd0 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_l ) / 100;  
629 :     }  
630 :  
631 :     /* 右後モータ */  
632 :     if( accele_r >= 0 ) {  
633 :         p2_3 = 0;  
634 :         trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * accele_r / 100;  
635 :     } else {  
636 :         p2_3 = 1;  
637 :         trdgrc1 = (long)( TRD_MOTOR_CYCLE - 2 ) * ( -accele_r ) / 100;  
638 :     }  
639 : }
```

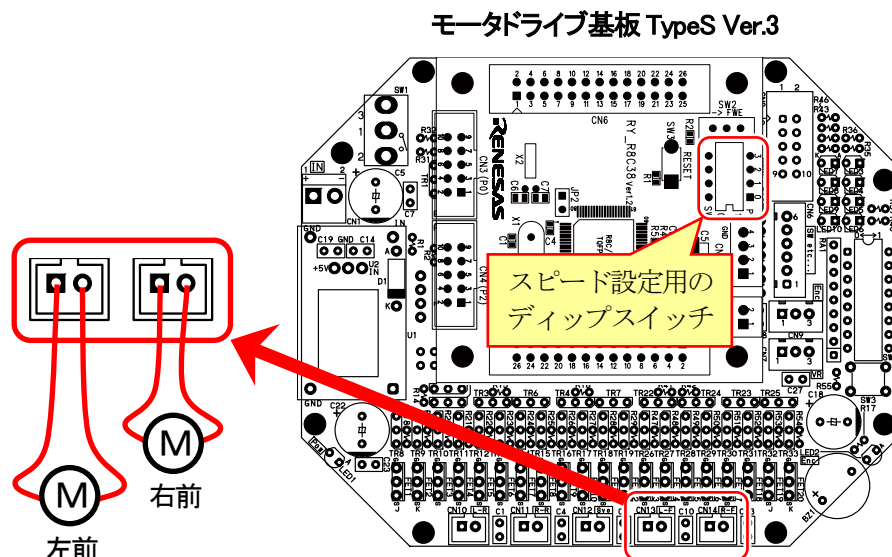
motor_r 関数は、ディップスイッチの割合でモータに出力する割合をさらに落としましたが、motor2_r 関数は、プログラムの引数どおりの割合をモータに出力します。

6. マイコンカー走行プログラムの解説

6.2.22 前輪の速度制御

```
641 : /*****  
642 : /* 前輪の速度制御 */  
643 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */  
644 : /*      0で停止、100で正転100%、-100で逆転100% */  
645 : /* 戻り値 なし */  
646 : /*****/  
647 : void motor_f( int accele_l, int accele_r )  
648 : {  
649 :     int sw_data;  
650 :  
651 :     sw_data = dipsw_get() + 5;          /* ディップスイッチ読み込み */  
652 :     accele_l = accele_l * sw_data / 20;  
653 :     accele_r = accele_r * sw_data / 20;  
654 :  
655 :     /* 左前モータ */  
656 :     if( accele_l >= 0 ) {  
657 :         p2_0 = 0;  
658 :     } else {  
659 :         p2_0 = 1;  
660 :         accele_l = -accele_l;  
661 :     }  
662 :     if( accele_l <= 5 ) {  
663 :         trcgrb = trcgrb_buff = trcgra;  
664 :     } else {  
665 :         trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_l / 100;  
666 :     }  
667 :  
668 :     /* 右前モータ */  
669 :     if( accele_r >= 0 ) {  
670 :         p2_7 = 0;  
671 :     } else {  
672 :         p2_7 = 1;  
673 :         accele_r = -accele_r;  
674 :     }  
675 :     if( accele_r <= 5 ) {  
676 :         trcgrd = trcgrd_buff = trcgra;  
677 :     } else {  
678 :         trcgrd_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_r / 100;  
679 :     }  
680 : }
```

モータドライブ基板 TypeS Ver.3 の前輪モータ 2 個を制御する関数です。



使い方を下記に示します。

```
motor_f ( 左前モータの PWM, 右前モータの PWM );
```

左前モータの PWM、右前モータの PWM の値は、下記を設定することができます。

- 0… 停止
- 1~100… 正転の割合 100 が一番速い
- 1~-100… 逆転の割合 100 が一番速い

モータへの出力は、motor_f関数で設定した割合がそのままモータに出力されるのではなく、下記の計算結果がモータに出力されます。

実際のモータに出力される割合 = 引数 × (マイコンボードのディップスイッチの値 + 5) ÷ 20

例えば、ディップスイッチの値が 12 で下記プログラムを実行したとします。

```
motor_f( 50, 100 );
```

左前モータに出力される割合 = プログラムの割合 × (ディップスイッチの値 + 5) ÷ 20
 = 50 × (12 + 5) ÷ 20
 = 42.5
 ≒ **42** (小数点以下は切り捨てです)

右前モータに出力される割合 = プログラムの割合 × (ディップスイッチの値 + 5) ÷ 20
 = 100 × (12 + 5) ÷ 20
 = **85**

6. マイコンカー走行プログラムの解説

6.2.23 前輪の速度制御 2 ディップスイッチには関係しない motor 関数

```
682 : /*****  
683 : /* 前輪の速度制御2 ディップスイッチには関係しないmotor関数 */  
684 : /* 引数 左モータ:-100~100 , 右モータ:-100~100 */  
685 : /*      0で停止、100で正転100%、-100で逆転100% */  
686 : /* 戻り値 なし */  
687 : /*****  
688 : void motor2_f( int accele_l, int accele_r )  
689 : {  
690 :     /* 左前モータ */  
691 :     if( accele_l >= 0 ) {  
692 :         p2_0 = 0;  
693 :     } else {  
694 :         p2_0 = 1;  
695 :         accele_l = -accele_l;  
696 :     }  
697 :     if( accele_l <= 5 ) {  
698 :         trcgrb = trcgrb_buff = trcgra;  
699 :     } else {  
700 :         trcgrb_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_l / 100;  
701 :     }  
702 :  
703 :     /* 右前モータ */  
704 :     if( accele_r >= 0 ) {  
705 :         p2_7 = 0;  
706 :     } else {  
707 :         p2_7 = 1;  
708 :         accele_r = -accele_r;  
709 :     }  
710 :     if( accele_r <= 5 ) {  
711 :         trcgrd = trcgrd_buff = trcgra;  
712 :     } else {  
713 :         trcgrd_buff = (unsigned long)(TRC_MOTOR_CYCLE-2) * accele_r / 100;  
714 :     }  
715 : }
```

motor_f 関数は、ディップスイッチの割合でモータに出力する割合をさらに落としましたが、motor2_f 関数は、プログラムの引数どおりの割合をモータに出力します。

6.2.24 後モータ停止動作(ブレーキ、フリー)設定

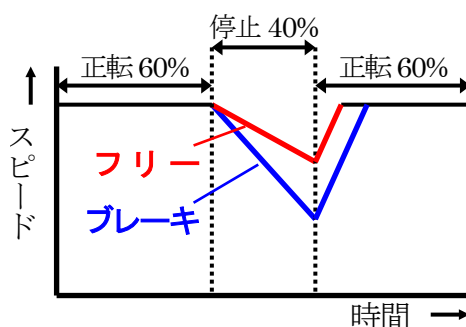
```

717 : /*****/
718 : /* 後モータ停止動作 (ブレーキ、フリー) */
719 : /* 引数 左モータ:FREE or BRAKE , 右モータ:FREE or BRAKE */
720 : /* 戻り値 なし */
721 : /*****/
722 : void motor_mode_r( int mode_l, int mode_r )
723 : {
724 :     if( mode_l ) {
725 :         p9_0 = 1;
726 :     } else {
727 :         p9_0 = 0;
728 :     }
729 :     if( mode_r ) {
730 :         p9_1 = 1;
731 :     } else {
732 :         p9_1 = 0;
733 :     }
734 : }

```

後モータを回すとき、停止の状態をブレーキにするかフリーにするか選択します。例えば、60%でモータを正転させるとき、60%が正転、残りの 40%が停止です。この 40%の停止状態をブレーキにするかフリーにするかを選択します。100%で回すときは、停止状態が無いいため、motor_mode_r 関数の設定は無効になります。

下図に停止時をブレーキにしたときとフリーにしたときのスピードイメージを示します。あくまでイメージです。実際はタイヤの抵抗など条件により変わりますので、各自検証してください。



6.2.25 前モータ停止動作(ブレーキ、フリー)設定

```

736 : /*****/
737 : /* 前モータ停止動作 (ブレーキ、フリー) */
738 : /* 引数 左モータ:FREE or BRAKE , 右モータ:FREE or BRAKE */
739 : /* 戻り値 なし */
740 : /*****/
741 : void motor_mode_f( int mode_l, int mode_r )
742 : {
743 :     if( mode_l ) {
744 :         p9_2 = 1;
745 :     } else {
746 :         p9_2 = 0;
747 :     }
748 :     if( mode_r ) {
749 :         p9_3 = 1;
750 :     } else {
751 :         p9_3 = 0;
752 :     }
753 : }

```

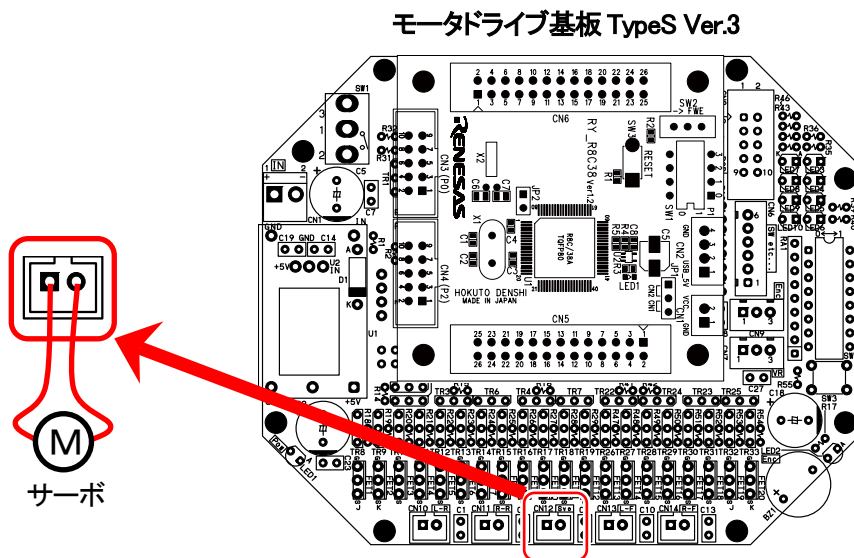
前モータを回すとき、停止の状態をブレーキにするかフリーにするか選択します。他は motor_mode_r 関数と同じです。

6.2.26 サーボモータの速度制御

```

755 : /*****/
756 : /* サーボモータ制御 */
757 : /* 引数 サーボモータPWM : -100~100 */
758 : /* 0で停止、100で正転100%、-100で逆転100% */
759 : /* 戻り値 なし */
760 : /*****/
761 : void servoPwmOut( int pwm )
762 : {
763 :     if( pwm >= 0 ) {
764 :         p2_6 = 0;
765 :         trdgrd1 = (long)( TRD_MOTOR_CYCLE - 2 ) * pwm / 100;
766 :     } else {
767 :         p2_6 = 1;
768 :         trdgrd1 = (long)( TRD_MOTOR_CYCLE- 2 ) * ( -pwm ) / 100;
769 :     }
770 : }
    
```

モータドライブ基板 TypeS Ver.3 のサーボモータを制御する関数です。



使い方は、下記のようになります。

```

servoPwmOut ( サーボモータの PWM );
    
```

サーボモータの PWM の値は、下記を設定することができます。

- 0… 停止
- 1~100… 右回転の割合 100 が一番速い
- 1~-100… 左回転の割合 100 が一番速い

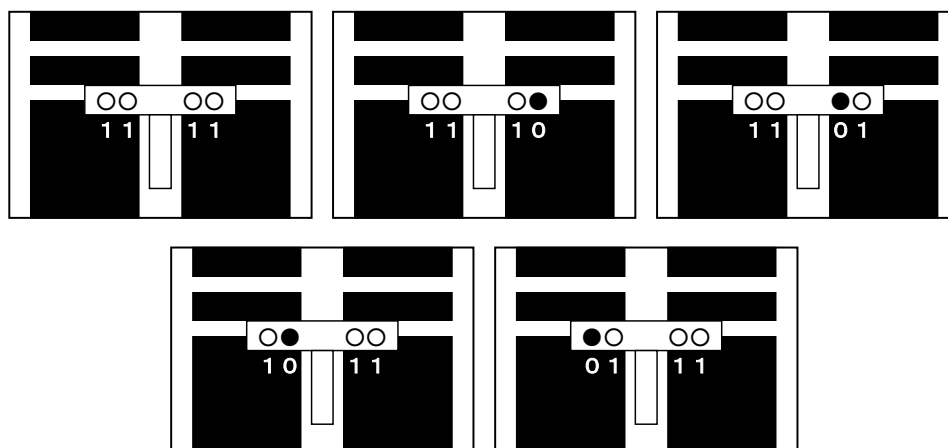
今回のサンプルプログラムは、引数を正の数にすると右へ、負の数にすると左へステアリングが回るよう配線します。逆の場合は、サーボモータの線を入れ変えてください。

6.2.27 クロスラインの検出処理

```

772 : /*****
773 : /* クロスライン検出処理
774 : /* 引数 なし
775 : /* 戻り値 0:クロスラインなし 1:あり
776 : /*****
777 : int check_crossline( void )
778 : {
779 :     unsigned char b;
780 :     int ret = 0;
781 :
782 :     b = sensor_inp();
783 :     if( b==0x0f || b==0x0e || b==0x0d || b==0x0b || b==0x07 ) {
784 :         ret = 1;
785 :     }
786 :     return ret;
787 : }
    
```

センサの状態をチェックして、クロスラインかどうか判断する関数です。アナログセンサ基板 TypeS Ver.2 の中心を除くデジタルセンサ 4 つの内、3 つ以上が白を検出するとクロスラインと判断します。戻り値は、クロスラインを検出したら"1"、クロスラインなしは"0"が返ってきます。



6. マイコンカー走行プログラムの解説

6.2.28 サーボモータ角度の取得

```

789 : /*****/
790 : /* サーボ角度取得 */
791 : /* 引数 なし */
792 : /* 戻り値 入れ替え後の値 */
793 : /*****/
794 : int getServoAngle( void )
795 : {
796 :     return( ad2 - iAngle0 );
797 : }
    
```

サーボモータの角度は、AN14(P7_2)端子に接続されているボリュームの値で分かります。iAngle0 は、0 度のときの A/D 変換値を入れておきます。例えば、角度が 0 度のとき A/D 変換値が 456 なら、iAngle0 変数に 456 を代入すると、戻り値は下記ようになります。

$$\begin{aligned}
 \text{戻り値} &= \text{A/D 変換値} - 0 \text{ 度のときの A/D 変換値} \\
 &= 456 - 456 \\
 &= 0
 \end{aligned}$$

今回のマイコンカーは左右 40 度ずつハンドルが切れました。中心と左右最大にハンドルを切ったときの電圧を測ります。テストでボリューム値を計った結果、

左いっぱい…3.26V 中心…2.23V 右いっぱい…1.21V

となりました(下左図)。5.00V が 1023 なので、それぞれの電圧を A/D 値に変換すると、

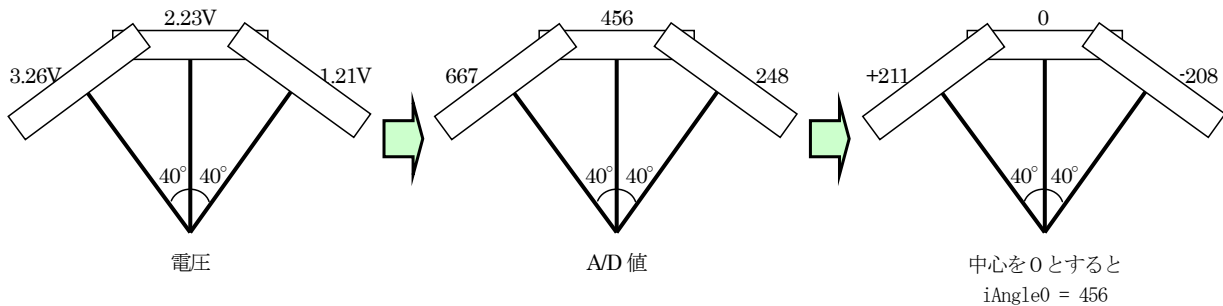
左いっぱい…3.26/5*1023=667 中心…2.23/5*1023=456 右いっぱい…1.21/5*1023=248

となります(下中図)。

796 行の iAngle0 変数には、0 度のときの A/D 値を入れておきます。iAngle0 変数に 456 の値を入れると、

左いっぱい…+211 中心…0 右いっぱい…-208

となります(下右図)。



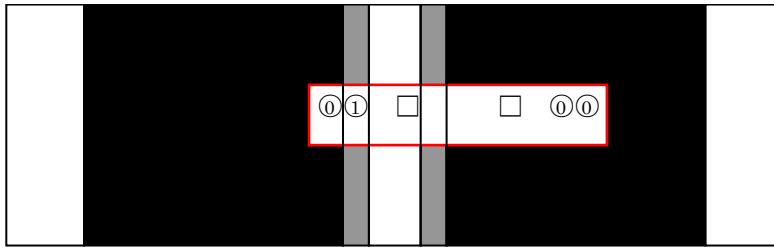
6.2.29 アナログセンサ値の取得

```
799 : /*****  
800 : /* アナログセンサ値取得 */  
801 : /* 引数 なし */  
802 : /* 戻り値 センサ値 */  
803 : /*****  
804 : int getAnalogSensor( void )  
805 : {  
806 :     int ret;  
807 :  
808 :     ret = ad1 - ad0;          /* アナログセンサ情報取得 */  
809 :  
810 :     if( !crank_mode ) {  
811 :         /* クランクモードでなければ補正処理 */  
812 :         switch( iSensorPattern ) {  
813 :             case 0:  
814 :                 if( sensor_inp() == 0x04 ) {  
815 :                     ret = -650;  
816 :                     break;  
817 :                 }  
818 :                 if( sensor_inp() == 0x02 ) {  
819 :                     ret = 650;  
820 :                     break;  
821 :                 }  
822 :                 if( sensor_inp() == 0x0c ) {  
823 :                     ret = -700;  
824 :                     iSensorPattern = 1;  
825 :                     break;  
826 :                 }  
827 :                 if( sensor_inp() == 0x03 ) {  
828 :                     ret = 700;  
829 :                     iSensorPattern = 2;  
830 :                     break;  
831 :                 }  
832 :                 break;  
833 :  
834 :             case 1:  
835 :                 /* センサ右寄り */  
836 :                 ret = -700;  
837 :                 if( sensor_inp() == 0x04 ) {  
838 :                     iSensorPattern = 0;  
839 :                 }  
840 :                 break;  
841 :  
842 :             case 2:  
843 :                 /* センサ左寄り */  
844 :                 ret = 700;  
845 :                 if( sensor_inp() == 0x02 ) {  
846 :                     iSensorPattern = 0;  
847 :                 }  
848 :                 break;  
849 :             }  
850 :         }  
851 :  
852 :         return ret;  
853 :     }
```

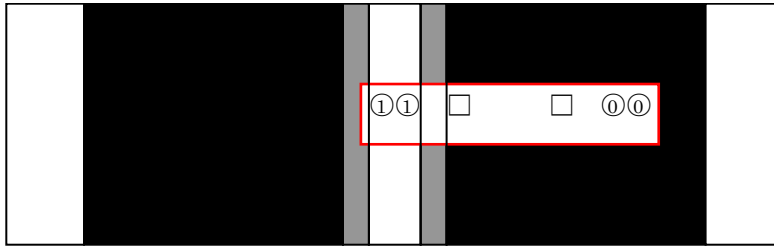
アナログセンサ左(P7_1 端子に接続)とアナログセンサ右(P7_0 端子に接続)の差分を計算して、コース中心からのずれを検出します。808 行で「アナログセンサ左－アナログセンサ右」の計算をしています。それ以降の行は、急

6. マイコンカー走行プログラムの解説

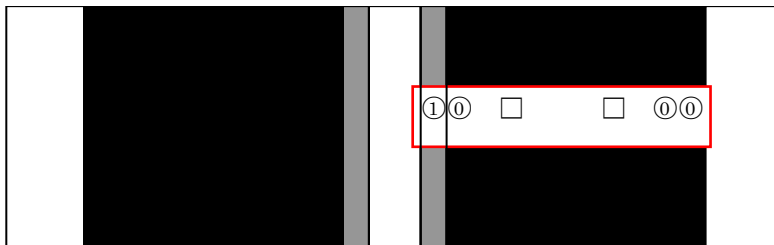
カーブのときにステアリングが反応しきれずにセンタラインを大きくはずれてしまった場合、デジタルセンサを使用して補正させる処理を行っています。その処理を 812 行から 849 行まで行っています。



センサが“0100”になると、センサ値を強制的に-650 とします。



デジタルセンサが“1100”になると、センサ値を強制的に-700 とします。**この状態をデジタルセンサが“0100”になるまで保持します。**



更にはずれてもデジタルセンサが“0100”になるまで-700 の値を保持し続けます。この状態になると、アナログセンサは両方とも黒色ですので差分をとっても 0 となります。補正がなければ、この状態を中心と認識してしまいます。

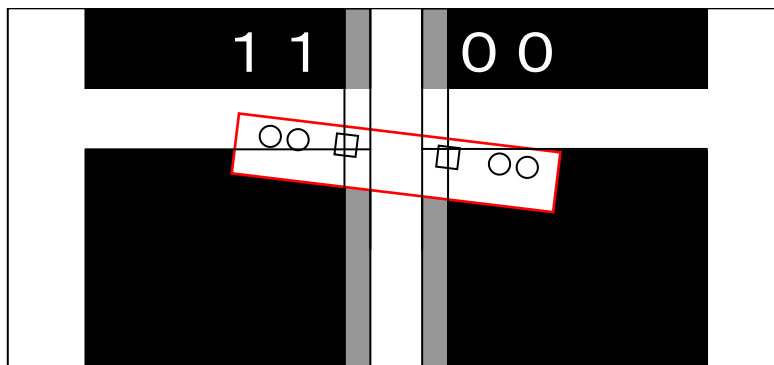
このように、アナログセンサだけでは追従しきれない場合を想定して、デジタルセンサを使いアナログセンサの値を補正しています。

逆のずれも、デジタルセンサの状態と値が変わるだけで考え方は同じです。

810 行目で crank_mode 変数の値をチェックしています。これは、クロスラインを検出したときや直角を検出するときにデジタルセンサの補正を行わないよう、補正機能を OFF するための変数です。crank_mode が 0 なら、if の { } 内は実行しません。

例えば、下図のようなときはクロスライン検出状態です。センサの反応は“1100”となり、補正するとセンサ値は-700 となってしまいます。もう少し進み、“1110”や“1111”になったらすぐに crank_mode を 1 として、デジタルセンサの補正を停止します。補正を停止しないと、急カーブと判断してしまい、ハンドルを切って脱輪します。

ちなみに、“1100”から“1111”に変化するまでの間は急カーブと判断してしまいますが、非常に短い時間なのでほとんど影響はありません。



6.2.30 サーボモータ制御

```
855 : /*****  
856 : /* サーボモータ制御 */  
857 : /* 引数 なし */  
858 : /* 戻り値 グローバル変数 iServoPwm に代入 */  
859 : /*****  
860 : void servoControl( void )  
861 : {  
862 :     int i, iRet, iP, iD;  
863 :     int kp, kd;  
864 :  
865 :     i = getAnalogSensor(); /* センサ値取得 */  
866 :     kp = dipsw_get2() & 0x0f; /* 調整できたら P,D 値は固定値に */  
867 :     kd = (dipsw_get2() >> 4) * 5; /* してください */  
868 :  
869 :     /* サーボモータ用 PWM 値計算 */  
870 :     iP = kp * i; /* 比例 */  
871 :     iD = kd * (iSensorBefore - i); /* 微分(目安は P の 5~10 倍) */  
872 :     iRet = iP - iD;  
873 :     iRet /= 64;  
874 :  
875 :     /* PWM の上限の設定 */  
876 :     if( iRet > 50 ) iRet = 50; /* マイコンカーが安定したら */  
877 :     if( iRet < -50 ) iRet = -50; /* 上限を 90 くらいにしてください */  
878 :     iServoPwm = iRet;  
879 :  
880 :     iSensorBefore = i; /* 次回はこの値が 1ms 前の値となる*/  
881 : }
```

この関数で、コースのセンターラインからセンサがどれだけずれているかを検出して、サーボモータの PWM 値を計算します。サーボ制御の要(かなめ)の部分です。

(1) PID 制御とは？

自動制御方式の中でもっとも良く使われる制御方式に PID 制御という方式があります。この PID とは

P : Proportional (比例)

I : Integral (積分)

D : Differential (微分)

の 3 つの組み合わせで制御する方式で、きめ細かくサーボモータの PWM を調整してスムーズな制御を行うことができます。

PID 制御についての詳細は、ホームページや書籍が多数出ていますのでそちらを参照してください。

今回、サーボモータの制御は比例制御と微分制御を行います。PD 制御と呼びます。

6. マイコンカー走行プログラムの解説

(2) P(比例)制御

比例制御とは、目標値からのずれに対して比例した制御量 P を与えます。P を計算する式は下記のようになります。

$$\text{制御量 } P = k_p \times p$$

k_p = 定数

p = 現在の値 - 目標の値

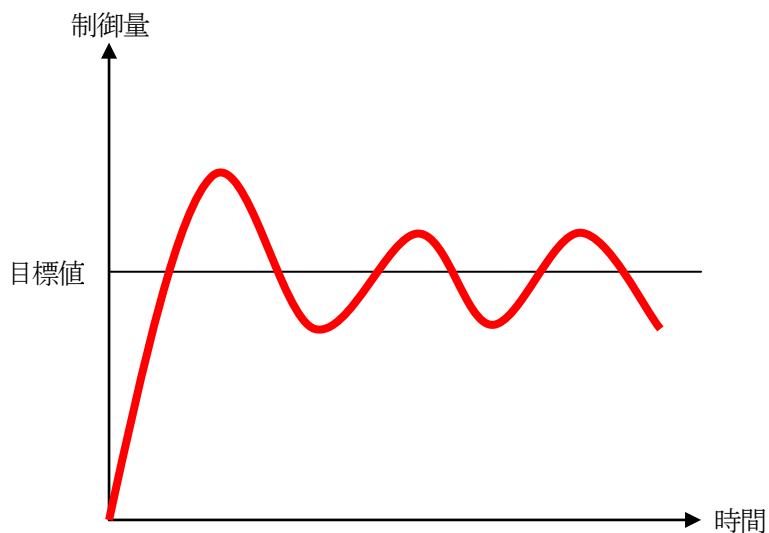
= 現在のアナログセンサ値 - 目標のアナログセンサ値

= `getAnalogSensor()` - 0

= `getAnalogSensor()`

目標のアナログセンサ値は、ちょうどコースの中心の値である 0 になります。

制御としては早く目標値に近づけたいので、ずれが大きいかほどサーボモータの PWM を多くします。そのため、目標値に到達しても速度を落としきれず、目標値をいたりきたりと振動してしまいます。



(3) D(微分)制御を加える

微分制御とは、瞬間的な変化量を計算して比例制御を押さえるような働きをします。微分制御量 D を計算する式は下記ようになります。

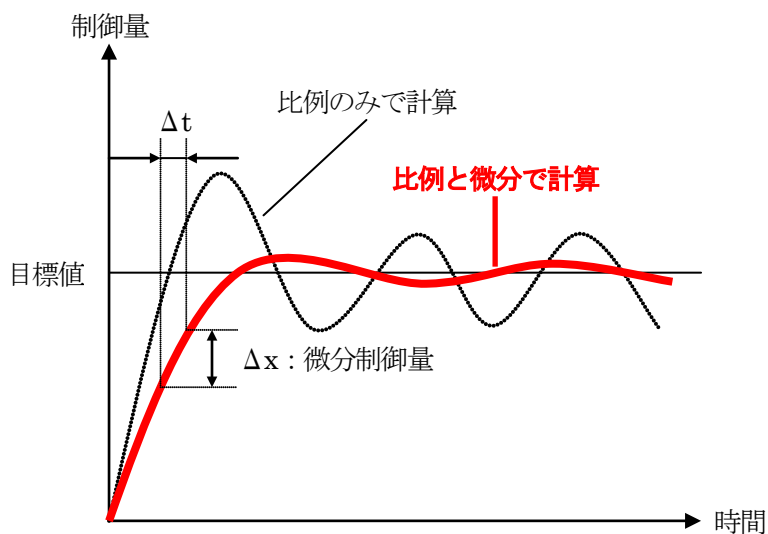
制御量 $D = kd \times d$

kd = 定数

d = 過去のアナログセンサ値 - 現在のアナログセンサ値
= $iSensorBefore$ - $getAnalogSensor()$

過去のアナログセンサ値を $iSensorBefore$ というグローバル変数に保存しておきます。

比例制御のみで振動していても、微分量を加えると振動を抑えることができます。ただし、比例制御を押さえる働きをしますので、目標値に近づく時間は長くなります。時間は数 ms~数十 ms のレベルです。それが、実際の走りに対して、どう影響するかは検証する必要があります。



(4) 最終制御量

サーボに加える制御量を計算する式は下記ようになります。

最終制御量 = P値 - D値

プログラムでは、最終制御量に定数をかけて PWM 値に調整します。

最後に、サーボモータに大きい PWM を加えるとステアリング部のギヤが壊れてしまうので、PWM の上限を設けます。サンプルプログラムは、50%以上にならないようにしています。この数値を小さくしすぎると、せつかくの PD 制御も上制限されてしまうので反応が遅くなります。大きすぎると万が一大きい PWM をかけてしまった場合、ギヤが壊れます。50%の設定は最初だけとして、コーストレースが安定したら 90%程度にしてください。

今回、

P制御の定数 = モータドライブ基板 TypeS Ver.3 のディップスイッチ下位 4 ビット

D制御の定数 = モータドライブ基板 TypeS Ver.3 のディップスイッチ上位 4 ビット

最終定数 = 1/64

としました。最初はディップスイッチの値をすべて 0 にしておきます。P 制御の定数、D 制御の定数は、モータ、ギヤ、電圧により違ってきますので個々のマイコンカーに合わせてカット&トライで調整する必要があります。調整するときは、1 つずつ値を増やしサーボがコースを滑らかにトレースするように調整してください。

6. マイコンカー走行プログラムの解説

6.2.31 内輪 PWM 値計算

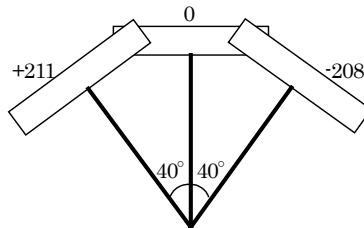
```

883 : /*****
884 : /* 外輪のPWMから、内輪のPWMを割り出す ハンドル角度は現在の値を使用 */
885 : /* 引数 外輪PWM */
886 : /* 戻り値 内輪PWM */
887 : *****/
888 : int diff( int pwm )
889 : {
890 :     int i, ret;
891 :
892 :     i = getServoAngle() / 5;          /* 1度あたりの増分で割る */
893 :     if( i < 0 ) i = -i;
894 :     if( i > 45 ) i = 45;
895 :     ret = revolution_difference[i] * pwm / 100;
896 :
897 :     return ret;
898 : }
    
```

各自のマイコンカーに
合わせて調整してください

diff関数は、引数に外輪(多く回るタイヤ側)のPWM 値を入れて呼び出すと、内輪(少なく回るタイヤ側)のPWM 値が返って来るとい関数です。

892 行で、現在の角度を取得します。getServoAngle 関数の戻り値は A/D 値なので、「度」に直す必要があります。ハンドルを±40 度動かしたときの A/D 値を、下記に示します。



左 40 度するとき A/D 値は 211、右 40 度するとき A/D 値は-208 でした。A/D 値や角度の測り方で若干の誤差がありますので、ここではそれぞれ±210 とします。A/D 値 210 のとき、左 40 度なので、1 度あたりの A/D 値は下記のようになります。

$$\begin{aligned}
 \text{1 度あたりの A/D 値} &= \text{左 40 度するときの A/D 値} \div \text{左 40 度} \\
 &= 210 \div 40 \\
 &= 5.25 \approx 5
 \end{aligned}$$

値は四捨五入して、整数にします。よって、A/D 値 5 が約 1 度となります。

893 行は、負の数を正の数に変換しています。

894 行は、45 度以上の角度のときは、45 度にしておきます。これは次に説明する配列の設定が、45 度までしか無いためです。

895 行は、左タイヤと右タイヤの回転差を計算します。まず、外輪の回転数を 100 と考えて、内輪の回転数を計算します。例えば、現在の角度が 25 度するとき、添え字部分に 25 が入り、内輪の回転数が戻り値となります。

```

ret = revolution_difference[i]
    = revolution_difference[ 25 ]
    = 65
    
```

外輪 100%のとき、内輪は戻り値である 65%であることが分かります。

次に、外輪が 100%で無い場合を計算します。内輪は外輪の回転に比例しますので、割合をかければ内輪の PWM 値が分かります。例えば、外輪が 60%なら内輪は次の計算で求めることができます。

```
ret = 65 * 外輪の PWM 値 / 100
     = 65 * 60 / 100
     = 39
```

サーボ角度が 25 度のとき、下記プログラムを実行すると kakudo 変数には 39 が代入されます。

```
kakudo = diff( 60 );
```

6.2.32 main 関数—初期化

```
105 : void main( void )
106 : {
107 :     int i;
108 :
109 :     /* マイコン機能の初期化 */
110 :     init();                               /* 初期化 */
111 :     asm(" fset I ");                       /* 全体の割り込み許可 */
112 :     initBeepS();                           /* ブザー関連処理 */
113 :
114 :     /* マイコンカーの状態初期化 */
115 :     motor_mode_f( BRAKE, BRAKE );
116 :     motor_mode_r( BRAKE, BRAKE );
117 :     motor_f( 0, 0 );
118 :     motor_r( 0, 0 );
119 :     servoPwmOut( 0 );
120 :     setBeepPatternS( 0x8000 );
```

main 関数では最初に、R8C/38A マイコンの内蔵周辺機能の初期化、割り込みの許可、ブザー関連処理の初期化、モータを停止状態にします。

6. マイコンカー走行プログラムの解説

6.2.33 パターン処理

マイコンカーの状態は pattern 変数で管理しています。通称、パターン処理と呼ぶことにします。pattern が 0 でスタート待ち、1 で通常トレースなど、それぞれの状態に応じてパターンを変えて処理内容を変えていきます。

現在のモード (pattern)	状態	pattern 変数が変わる条件
0	プッシュスイッチ押下待ち	・プッシュスイッチを押したら 1 へ
1	スタートバー開待ち	・スタートバーが開いたら 11 へ
11	通常トレース	・クロスラインを検出したら 21 へ
21	クロスライン通過処理	・200ms たったら 22 へ
22	クロスライン後のトレース、直角検出処理	・右クランクを見つけたら 31 へ ・左クランクを見つけたら 41 へ
31	右クランク処理	・曲げ終わりを検出すると 32 へ
32	少し時間がたつまで待つ	・100ms たったら 11 へ
41	左クランク処理	・曲げ終わりを検出すると 42 へ
42	少し時間がたつまで待つ	・100ms たったら 11 へ
その他	—	・0 へ

6.2.34 パターン 0:スタート待ち

```

122 :   while( 1 ) {
123 :
124 :     switch( pattern ) {
125 :     case 0:
126 :       /* プッシュスイッチ押下待ち */
127 :       servoPwmOut( 0 );
128 :       if( pushsw_get() ) {
129 :         setBeepPatternS( 0xcc00 );
130 :         cnt1 = 0;
131 :         pattern = 1;
132 :         break;
133 :       }
134 :       i = (cnt1/200) % 2 + 1;
135 :       if( startbar_get() ) {
136 :         i += ((cnt1/100) % 2 + 1) << 2;
137 :       }
138 :       led_out( i );           /* LED 点滅処理          */
139 :       break;

```

プッシュスイッチ押下待ちです。プッシュスイッチを押すまでの間、LED8 個中 2 個を点滅させプッシュスイッチが押されるまで待ちます。また、スタートバー検出センサが反応するとさらに 2 個(合計 4 個)点滅させ、スタートバー閉を検出していることを、選手に分かりやすく知らせます。

プッシュスイッチを押すと、下記処理を実行します。

- ブザーを鳴らす(ボタンを押した確認)
- パターン 1 へ移ります。

6.2.35 パターン 1:スタートバー開待ち

```
141 :     case 1:
142 :         /* スタートバー開待ち */
143 :         servoPwmOut( iServoPwm / 2 );
144 :         if( !startbar_get() ) {
145 :             iAngle0 = getServoAngle(); /* 0度の位置記憶          */
146 :             led_out( 0x0 );
147 :             cnt1 = 0;
148 :             pattern = 11;
149 :             break;
150 :         }
151 :         led_out( 1 << (cnt1/50) % 4 );
152 :         break;
```

パターン 1 は、スタートバーが開かれるのを待っている状態です。

143 行でサーボ制御を行っています。iServoPwm 変数がサーボモータに加える PWM 値です。割り込みプログラム内で 1ms ごとに自動で更新されていきます。スタート時、センサがブルブル震えないように、PWM 値を半分にしていきます。

スタートバーが開かれると、現在の角度を getServoAngle 関数で読み込み、その値を iAngle0 変数にセットして、**この状態を 0 度とします**。その後パターン 11 に移行します。

6. マイコンカー走行プログラムの解説

6.2.36 パターン 11: 通常トレース

```

154 :     case 11:
155 :         /* 通常トレース */
156 :         servoPwmOut( iServoPwm );
157 :         i = getServoAngle();
158 :         if( i > 170 ) {
159 :             motor_f( 0, 0 );
160 :             motor_r( 0, 0 );
161 :         } else if( i > 25 ) {
162 :             motor_f( diff(80), 80 );
163 :             motor_r( diff(80), 80 );
164 :         } else if( i < -170 ) {
165 :             motor_f( 0, 0 );
166 :             motor_r( 0, 0 );
167 :         } else if( i < -25 ) {
168 :             motor_f( 80, diff(80) );
169 :             motor_r( 80, diff(80) );
170 :         } else {
171 :             motor_f( 100, 100 );
172 :             motor_r( 100, 100 );
173 :         }
174 :         if( check_crossline() ) {           /* クロスラインチェック           */
175 :             cnt1 = 0;
176 :             crank_mode = 1;
177 :             pattern = 21;
178 :         }
179 :         break;

```

156 行でサーボ制御を行っています。次に 157 行でハンドル角度を取得します。角度に応じて左右回転数の設定をしています。サンプルプログラムは、ハンドル角度と駆動モータの関係を下記のようにします。

A/D 値	角度に変換 A/D 値÷5	左モータ PWM	右モータ PWM
171 以上	34 以上	0	0
26~170	5~34	diff(80)	80
-171 以下	-34 以下	0	0
-26~-170	-5~-34	80	diff(80)
それ以外 (-25~25)	-5~5	100	100

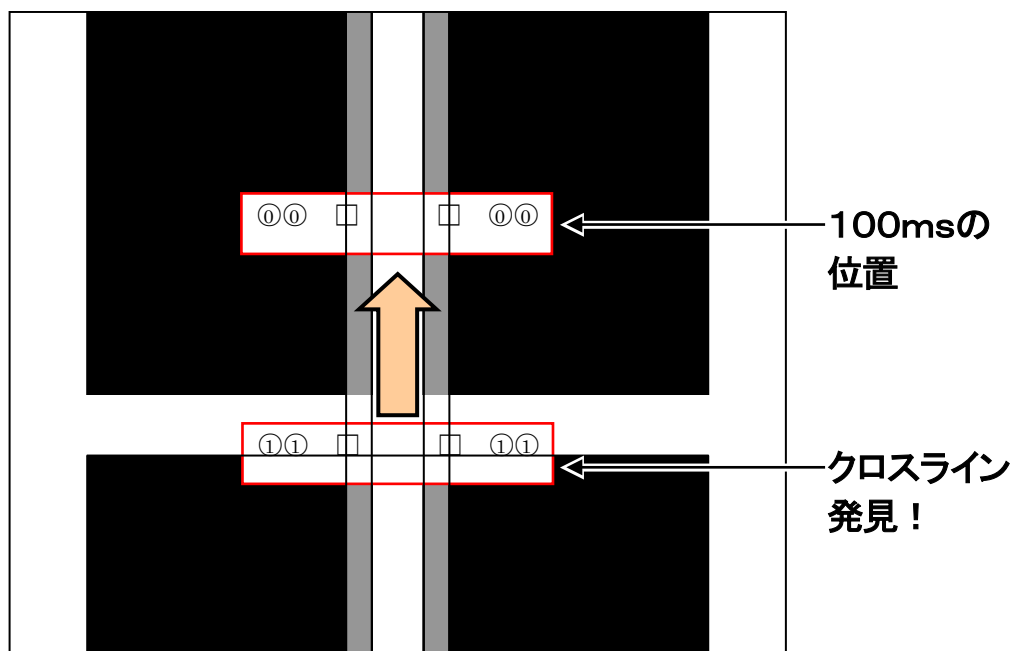
A/D 値が、±(26~170)なら、外輪を 80%として、内輪をステアリングの切れ角に応じて PWM 値を可変します。最後に、174 行でクロスラインチェックを行います。クロスラインを検出すると crank_mode に 1 を代入して、アナログセンサ値を取得する getAnalogSensor 関数内でデジタルセンサ補正を行わないようにします。パターンは 21 へ移行します。

6.2.37 パターン 21:クロスライン検出処理

```

181 :     case 21:
182 :         /* クロスライン通過処理 */
183 :         servoPwmOut( iServoPwm );
184 :         led_out( 0x3 );
185 :         motor_f( 0, 0 );
186 :         motor_r( 0, 0 );
187 :         if( cnt1 >= 100 ) {
188 :             cnt1 = 0;
189 :             pattern = 22;
190 :         }
191 :         break;
    
```

ここではブレーキをかけて、サーボ制御を行います。100ms の間にクロスラインを通過させ、100ms 後にはパターン 22 へ移行します。クロスラインを通過しきる前にパターン 22 に移ってしまうと、クロスラインを直角と見間違っ脱輪してしまいます。



6. マイコンカー走行プログラムの解説

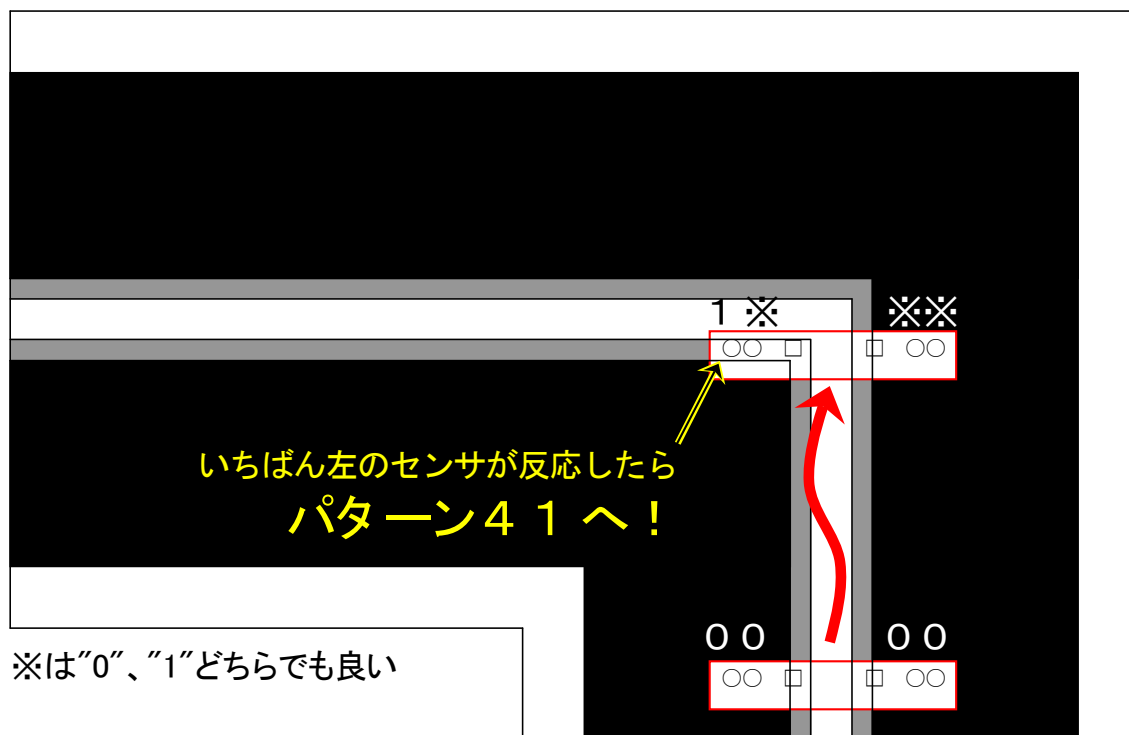
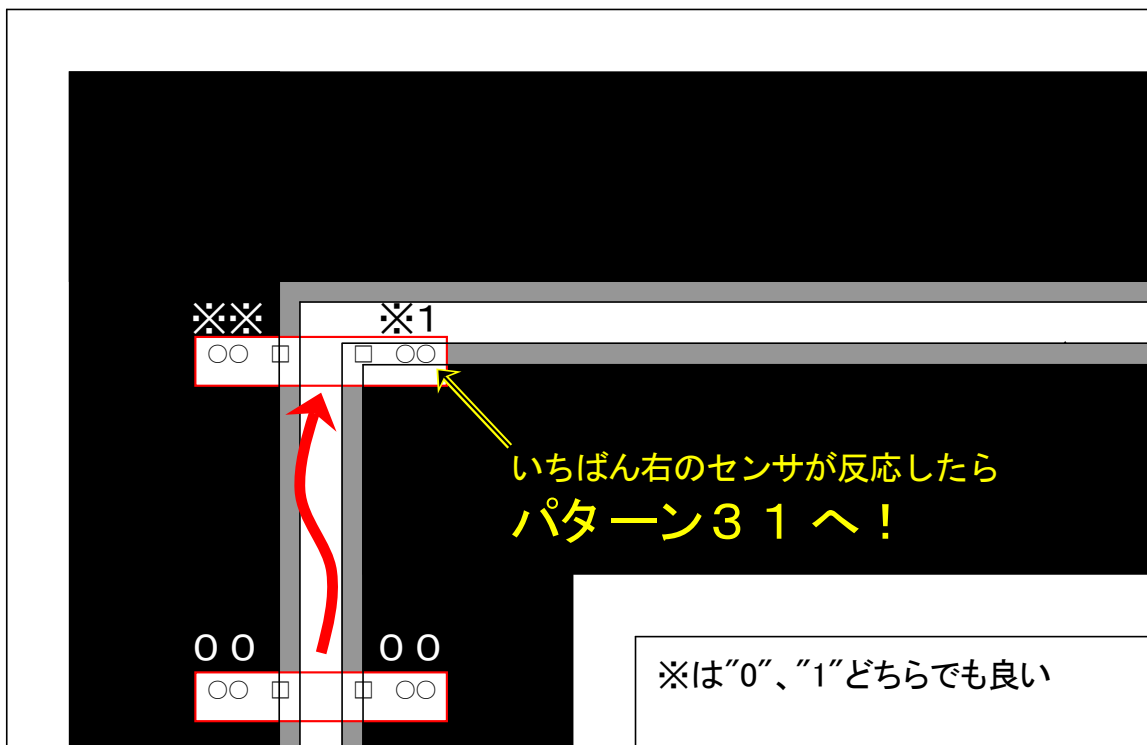
6.2.38 パターン 22:クロスライン後のトレース、直角検出処理

```
193 :     case 22:
194 :         /* クロスライン後のトレース、直角検出処理 */
195 :         servoPwmOut( iServoPwm );
196 :         if( iEncoder >= 11 ) {           /* エンコーダによりスピード制御 */
197 :             motor_f( 0, 0 );
198 :             motor_r( 0, 0 );
199 :         } else {
200 :             motor2_f( 70, 70 );
201 :             motor2_r( 70, 70 );
202 :         }
203 :
204 :         if( (sensor_inp()&0x01) == 0x01 ) { /* 右クランク?           */
205 :             led_out( 0x1 );
206 :             cnt1 = 0;
207 :             pattern = 31;
208 :             break;
209 :         }
210 :         if( (sensor_inp()&0x08) == 0x08 ) { /* 左クランク?           */
211 :             led_out( 0x2 );
212 :             cnt1 = 0;
213 :             pattern = 41;
214 :             break;
215 :         }
216 :         break;
```

クロスライン通過後の処理を行います。

196～202 行でロータリエンコーダによる速度制御を行っています。サンプルプログラムは、1m/s 以上なら PWM0%、以下なら PWM70%で走行します。

204 行目で、いちばん右のデジタルセンサのみをチェック、反応すれば右クランクと判断しパターン 31 へ移ります。同様に 210 行目で、いちばん左のセンサのみをチェック、反応すれば左クランクと判断しパターン 41 へ移動します。



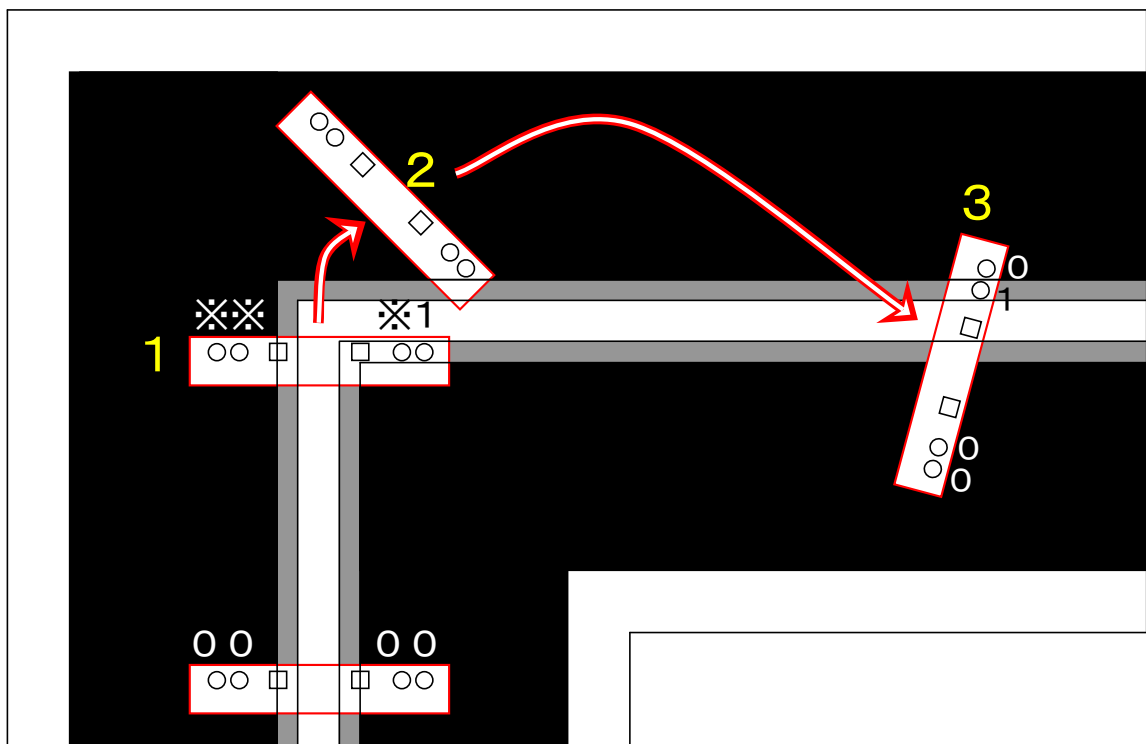
6. マイコンカー走行プログラムの解説

6.2.39 パターン 31: 右クランク処理

```

218 :     case 31:
219 :         /* 右クランク処理 */
220 :         servoPwmOut( 50 );           /* 振りが弱いときは大きくする */
221 :         motor_f( 60, 33 );          /* この部分は「角度計算(4WD時).xls」 */
222 :         motor_r( 49, 22 );          /* で計算 */
223 :         if( sensor_inp() == 0x04 ) { /* 曲げ終わりチェック */
224 :             cnt1 = 0;
225 :             iSensorPattern = 0;
226 :             crank_mode = 0;
227 :             pattern = 32;
228 :         }
229 :         break;
    
```

パターン 31 は、右にハンドルを曲げて、曲げ終わりかどうかチェックしている状態です。
 サーボモータのPWMはセンサ状態に関係なく右に50%回転させています。サーボの動きが遅い場合はこの値を大きくしますが、曲げすぎて車体にステアリング部分がぶつかりロックしないようにしてください。サーボモータのトルクが大きすぎたりギヤが弱い場合、ギヤがかかけたり車体が曲がったりすることがありますので気をつけます。
 何処まで回し続けるかというのが223行です。中心以外のデジタルセンサをチェックして、“0100”ならパターン32に移ります。移る前に、クランクが終わったので crank_mode 変数を0に戻します。



1. いちばん右のデジタルセンサが“1”になったので、右クランクと判断しサーボモータを50%、左前モータを60%、右前モータを33%、左後モータを49%、右後モータを22%で回します。左右回転差は、ハンドル40度で計算しています。
2. デジタルセンサが“0100”になるまで待ちます。まだです。
3. デジタルセンサが“0100”になりました。パターン32へ移ります。

6.2.40 パターン 32: 右クランク処理後、少し時間がたつまで待つ

```
231 :     case 32:
232 :         /* 少し時間が経つまで待つ */
233 :         servoPwmOut( iServoPwm );
234 :         motor2_r( 80, 80 );
235 :         motor2_f( 80, 80 );
236 :         if( cnt1 >= 100 ) {
237 :             led_out( 0x0 );
238 :             pattern = 11;
239 :         }
240 :         break;
```

右クランク処理終了後、100ms 間は駆動モータを 80%にします。これはパターン 32 に移ってきたときは、ハンドルをかなり曲げています。この状態でパターン 11 に戻ると、ボリューム値が-170 以下なのでモータスピードが左右共に 0%になってしまいます。これを防ぐために 100ms 間、ハンドルの角度に関係なく PWM を 80%にして少し進ませます。

6.2.41 パターン 41: 左クランク処理

```
242 :     case 41:
243 :         /* 左クランク処理 */
244 :         servoPwmOut( -50 );           /* 振りが弱いときは大きくする */
245 :         motor_f( 33, 60 );           /* この部分は「角度計算(4WD時).xls」 */
246 :         motor_r( 22, 49 );           /* で計算 */
247 :         if( sensor_inp() == 0x02 ) { /* 曲げ終わりチェック */
248 :             cnt1 = 0;
249 :             iSensorPattern = 0;
250 :             crank_mode = 0;
251 :             pattern = 42;
252 :         }
253 :         break;
```

左クランクも同様です。サーボモータは左へ 50%で回転させ、駆動モータを 40 度ハンドルを切ったと仮定して PWM を設定します。この状態をデジタルセンサの状態が"0010"になるまで繰り返します。"0010"になるとパターン 42 へ移ります。

6.2.42 パターン 42: 左クランク処理後、少し時間がたつまで待つ

```
255 :     case 42:
256 :         /* 少し時間が経つまで待つ */
257 :         servoPwmOut( iServoPwm );
258 :         motor2_f( 80, 80 );
259 :         motor2_r( 80, 80 );
260 :         if( cnt1 >= 100 ) {
261 :             led_out( 0x0 );
262 :             pattern = 11;
263 :         }
264 :         break;
```

左クランク処理終了後、100ms 間は駆動モータを 80%にします。これはパターン 42 に移ってきたときは、ハンドル

6. マイコンカー走行プログラムの解説

をかなり曲げています。この状態でパターン 11 に戻ると、ボリューム値が 170 以下なのでモータスピードが左右共に 0%になってしまいます。これを防ぐために 100ms 間、ハンドルの角度に関係なく PWM を 80%にして少し進ませます。

6.3 ブザー制御プログラムの解説

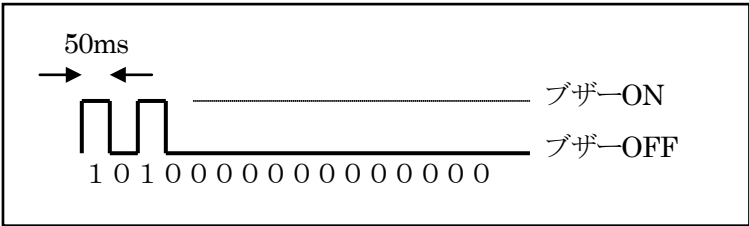
「types3_beep.c」は、モータドライブ基板 TypeS Ver.3 のブザーを制御する専用の関数が用意されているファイルです。モータドライブ基板 TypeS Ver.3 のブザーを使用するときは、プロジェクトに「types3_beep.c」を追加して使用します。

このファイルを追加したときに実行できる関数を説明します。

6.3.1 ブザー関連変数の初期化

書式	void initBeepS(void)		
内容	「types3_beep.c」ファイル内の変数を初期化します。		
引数	なし		
戻り値	なし		
使用例	<pre>init(); asm(" fset I "); initBeepS();</pre>	<pre>/* 初期化 */ /* 全体の割り込み許可 */ /* ブザー関連処理 */</pre>	<pre>*/ */ */</pre>

6.3.2 ブザーの出力パターンセット

書式	void setBeepPatternS(unsigned int data)
内容	ブザー出力パターンをセットします。
引数	<p>ブザー出力パターンを設定します。 値は 16 ビット分指定します。1 ビットあたり 50 ミリ秒の長さの音を鳴らします。例えば 16 進数で「0xa000」を設定したとします。2進数に直すと「1010 0000 0000 0000」となり、これは 50ms ブザー ON、50ms ブザー OFF、50ms ブザー ON、残りブザー OFF という設定です。耳には、「ピッピッ」と聞こえます。</p>  <p>もし、「0x8000」にすると、50ms ブザーが ON、後は全て OFF、「0xffff」なら、50ms×16=800ms 間ブザーが鳴り続けます。</p>
戻り値	なし
使用例	<pre> setBeepPatternS(0x8000); // ピッ setBeepPatternS(0xc000); // ピーッ setBeepPatternS(0xf000); // ピーーッ setBeepPatternS(0xa000); // ピッピッ setBeepPatternS(0xaaaa); // ピッピッピッピッピッピッピッピッ (8回) </pre>

6.3.3 ブザー処理

書式	void beepProcessS(void)
内容	ブザーを実際に鳴らす処理です。ブザー処理を行います。1ms ごとにこの関数を実行してください。
引数	なし
戻り値	なし
使用例	<pre> #pragma interrupt /B intTRB(vect=24) void intTRB(void) //タイマ RB 割り込み処理 { /* ブザー処理 */ beepProcessS(); // 1ms ごとに実行 } </pre>

7. 調整のポイント

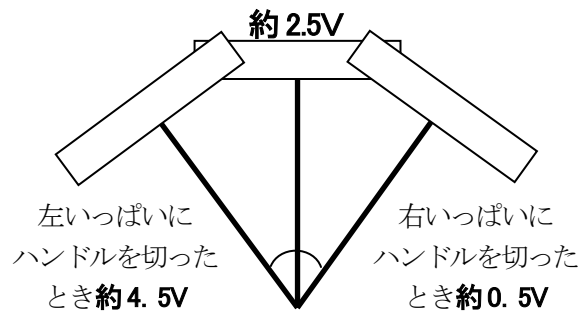
7.1 サーボモータの回転方向

サーボモータを接続するコネクタの 2 ピン側のモータ端子に+、1 ピン側のモータ端子に-の電圧を加えたとき、進行方向向かって右側にステアリングが回転するようにします。逆の場合、モータの線を左右入れ替えます。

7.2 ボリュームの調整

中心はほぼ 2.5V になるようにボリュームの向きとステアリングの角度を合わせます。

左右それぞれいっぱいまでハンドルを切ったとき、 $2.5 \pm 2V$ になるようにするのが理想です。A/D 値をいっぱいまで使用した方が、ちょっとのハンドルの曲げでも数値が変化するので精度が良くなります。一応下限の 0V と上限の 5V まで 0.5V の余裕を持たせて、切りすぎたときに変化しないということがないようにしています。下図に、その様子を示します。



左にハンドルを切ったときに電圧が高くなるように、右は電圧が低くなるように配線します。逆の場合は、ボリュームの 1 ピンと 3 ピンの線を逆にします。

ギヤの関係で、電圧の変換範囲が小さくなくても構いませんが精度が悪くなります。今回の説明用マイコンカーはギヤ比の関係で約 $\pm 1.0V$ しか電圧が変化しません。これは悪い例です。

7.3 角度を測っておく

最大まで曲げたときの、角度を測っておきます。また、いっしょにそのときの A/D 値も計算しておきます。

説明用マイコンカーの場合、テストでボリュームの電圧を計った結果、下記のようにになりました。

左いっぱい…3.26V 中心…2.23V 右いっぱい…1.21V

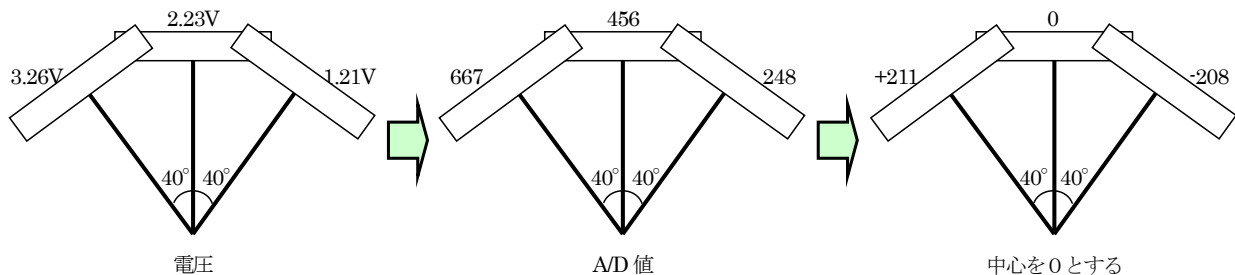
5.00V が 1023 なので、それぞれの電圧を A/D 値に変換すると、下記のようになります。

左いっぱい… $3.26/5 \times 1023 = 667$ 中心… $2.23/5 \times 1023 = 456$ 右いっぱい… $1.21/5 \times 1023 = 248$

中心を 0 とすると、下記のようになります。

左いっぱい…先ほどの計算結果－中心値＝667－456＝ 211
右いっぱい…先ほどの計算結果－中心値＝248－456＝－208

分度器で最大曲げ角を測ると、左右に 40 度ずつ曲げることができました。これらをまとめると下図のようになります。



ちなみに、1度あたりの A/D 値は、 $211/40 \approx 5$ です。
同様に、自分のマイコンカーの値を計算しておきましょう。

7.4 プログラムの調整のポイント

このサンプルプログラムは、説明用マイコンカーの車体に合わせて作成しています。もし、このサンプルプログラムを使用する場合は、自分のマイコンカーに合わせる必要があります。そのポイントを解説します。

行	内容	説明
90～100	内輪の PWM 値	エクセルの「角度計算.xls」を使用します。ホイールベースとトレッドを自分のマイコンカーの長さに合わせて入力し、配列の値を更新してください。
158	左へハンドルを切ったとき PWM を 0 にするときの A/D 値	最大まで左へハンドルを切ったときの約 8 割の値を使用します。説明用マイコンカーは 211 だったので $211 \times 0.8 = 168.8 \approx 170$ を設定します。
161	左へ 5 度ほどハンドルを切ったときの A/D 値	説明用マイコンカーは 1 度あたり A/D 値は 5 なので、5 度のときの A/D 値は、 $5 \times 5 = 25$ を設定します。
162,163	左へ 5 度以上ハンドルを切ったときの PWM 値	説明用マイコンカーは 80% ですが、それぞれのマイコンカーに合わせてください。
164	右へハンドルを切ったとき PWM を 0 にするときの A/D 値	最大まで右へハンドルを切ったときの約 8 割の値を使用します。説明用マイコンカーは -208 だったので $-208 \times 0.8 = -166.4 \approx -170$ を設定します。
167	右へ 5 度ほどハンドルを切ったときの A/D 値	説明用マイコンカーは 1 度あたり A/D 値は 5 なので、5 度のときの A/D 値は、 $5 \times -5 = -25$ を設定します。
168,169	右へ 5 度以上ハンドルを切ったときの PWM 値	説明用マイコンカーは 80% ですが、それぞれのマイコンカーに合わせてください。

7. 調整のポイント

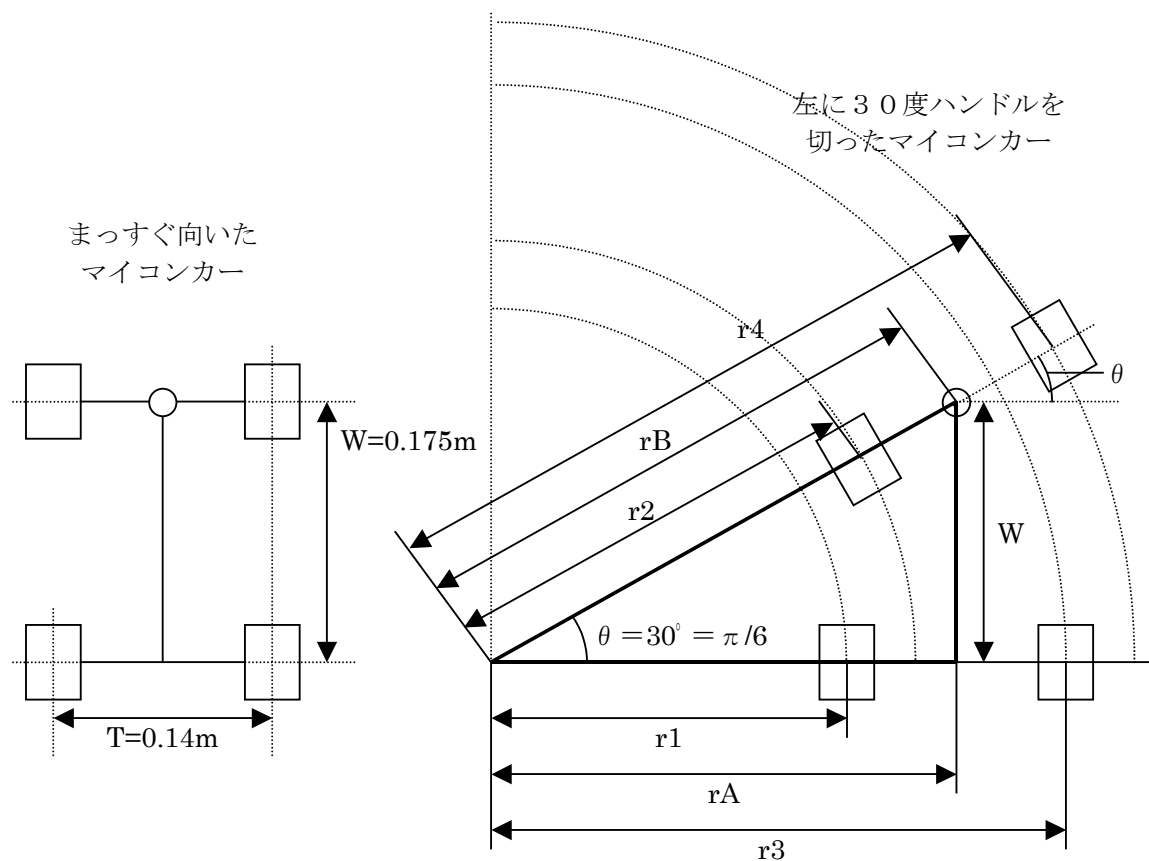
196	クロスライン検出後の スピード	あらかじめ 1m/s で進んでいるときの 10ms ごとのパルス値を計算しておきます。説明用マイコンカーはロータリエンコーダ製作キット Ver.2 を使用しているので 10.92 です。 サンプルプログラムは約 1m/s で走行させています。整数しか使えないので数値は四捨五入します。今回は 11 になります。 それぞれのマイコンカーのクランクを曲がれるスピードに設定します。もし自分のマイコンカーが 2m/s でクランクを曲がれるなら、 $10.92 \times 2 = 21.84 \approx 22$ となります。
220	右クランク検出時の ハンドルを曲げる PWM 値	右クランク検出時、右にハンドルを曲げる PWM 値を設定します。今回は 50% です。右に曲げ続けると車体にステアリングがぶつかりロックしてしまいます。かといって小さすぎると、曲げるスピードが遅くなり脱輪の原因となります。カット&トライで調整してください。
221,222	右クランククリア時の PWM 値	右クランククリア時の PWM 値を設定します。説明用マイコンカーでは外輪を 60%、ハンドル角度を 40 度と仮定して、「角度計算(4WD 時).xls」で計算した値にします。右に曲がるので、右側のモータが外輪になります。
244	左クランク検出時の ハンドルを曲げる PWM 値	左クランク検出時、左にハンドルを曲げる PWM 値を設定します。今回は -50% です。左に曲げ続けると車体にステアリングがぶつかりロックしてしまいます。かといって小さすぎると、曲げるスピードが遅くなり脱輪の原因となります。カット&トライで調整してください。
245,246	左クランククリア時の PWM 値	左クランククリア時の PWM 値を設定します。サンプルプログラムでは外輪を 60%、ハンドル角度を 40 度と仮定して、「角度計算(4WD 時).xls」で計算した値にします。左に曲がるので、左側のモータが外輪になります。
866	サーボモータ PD 制御の比例定数	中心線からのずれに応じてサーボモータを回す強さです。この値は、モータドライブ基板 TypeS Ver.3 のディップスイッチ 下位 4 ビットで 0~15 まで設定することができます。カット&トライで調整し、値が分かたら固定値にしてください。
867	サーボモータ PD 制御の微分定数	比例定数が小さすぎると中心線からずれたときの戻りが遅くなります。かといって大きすぎると、ハンドルが左右にブルブル震えて発振してしまいます。この微分定数を加えることにより、ブルブルを押さえることができます。 この値はモータドライブ基板 TypeS Ver.3 のディップスイッチ 上位 4 ビットで 0~15 まで設定することができます。カット&トライで調整し、値が分かたら固定値にしてください。
876,877	サーボモータに加える PWM の上限設定	サーボモータに加える PWM の上限を設定しています。サンプルプログラムは、50% 以上にならないようにしています。この数値が小さくしすぎると、せつかくの PD 制御もこの部分で制限されてしまうので反応が遅くなります。大きすぎると万が一大きい PWM をかけてしまった場合、モータやギヤなどが壊れます。最初は 50% として、ライントレースが安定したら 90% 程度にしてください。 例) 876 : if(iRet > 90) iRet = 90; 877 : if(iRet < -90) iRet = -90;

8. 4 輪の回転数計算

「角度計算(4WD 時).xls」で、下記計算ができます。

8.1 センターピボット方式 4 輪の回転数計算

センターピボット方式の 4 輪の回転数の計算方法を説明します。



T=トレッド…左右輪の中心線の距離 (キットでは 0.17[m]です)

W=ホイールベース…前輪と後輪の間隔 (キットでは 0.17[m]です)

図のように、後輪部の底辺 r_A 、高さ W 、角度 θ の三角形の関係は次のようです。

$$\tan \theta = W / r_A$$

角度 θ 、 W が分かっていますので、 r_A が分かります。

$$r_A = W / \tan \theta = 0.175 / \tan(\pi / 6) = 0.303[\text{m}]$$

後輪内輪の半径は、

$$r_1 = r_A - T/2 = 0.303 - 0.07 = 0.233[\text{m}]$$

8. 4 輪の回転数計算

後輪外輪の半径は、

$$r_3 = r_A + T/2 = 0.303 + 0.07 = 0.373[\text{m}]$$

また、前輪部の底辺 r_B 、高さ W 、角度 θ の三角形の関係は次のようです。

$$\sin \theta = W / r_B$$

角度 θ 、 W が分かっていますので、 r_B が分かります。

$$r_B = W / \sin \theta = 0.175 / \sin(\pi/6) = 0.350[\text{m}]$$

前輪内輪の半径は、

$$r_2 = r_B - T/2 = 0.350 - 0.07 = 0.280[\text{m}]$$

前輪外輪の半径は、

$$r_4 = r_B + T/2 = 0.350 + 0.07 = 0.420[\text{m}]$$

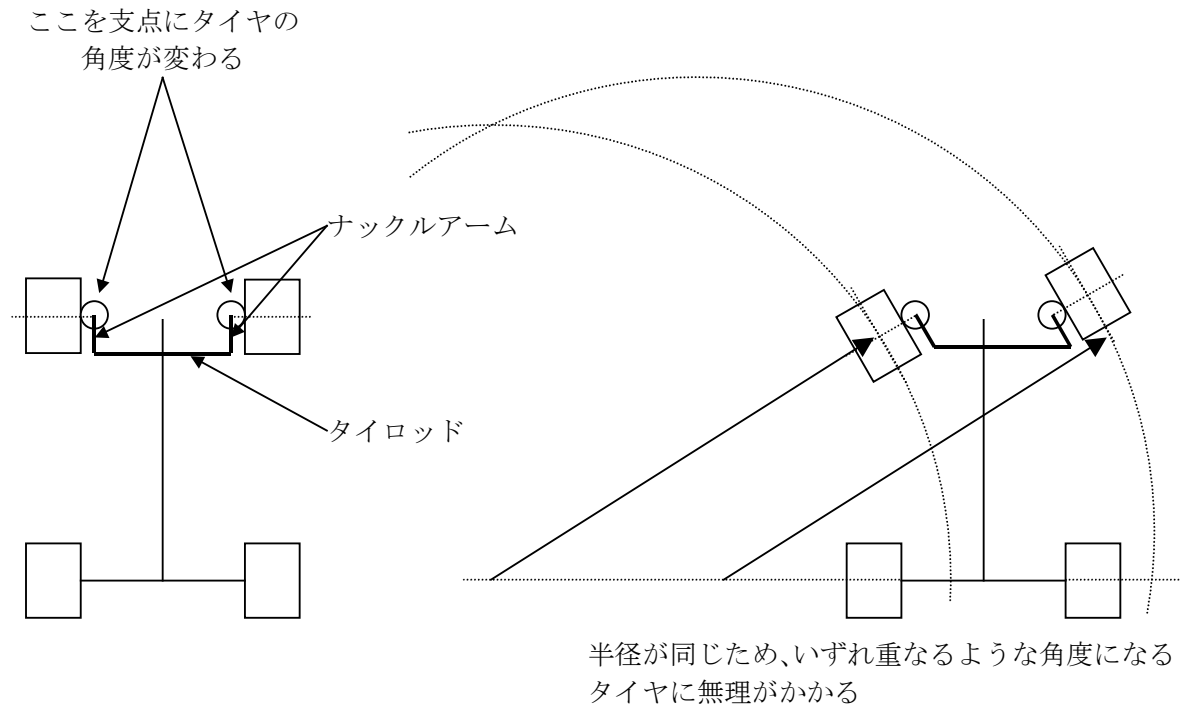
一番回転する r_4 を 100 としたときのそれぞれの回転数は、

$$\begin{aligned} r_1 : r_2 : r_3 : r_4 \\ &= 0.233 : 0.280 : 0.373 : 0.420 \\ &= 0.233 \times 100 / 0.420 : 0.280 \times 100 / 0.420 : 0.373 \times 100 / 0.420 : 0.420 \times 100 / 0.420 \\ &= 55 : 67 : 89 : 100 \end{aligned}$$

ハンドル角度 30 度、前輪外輪が 100% で回転するとき、後輪外輪は 89 回転、前輪内輪は 67 回転、後輪内輪は 55 回転することになります。

8.2 アッカーマン方式 4 輪の回転数計算

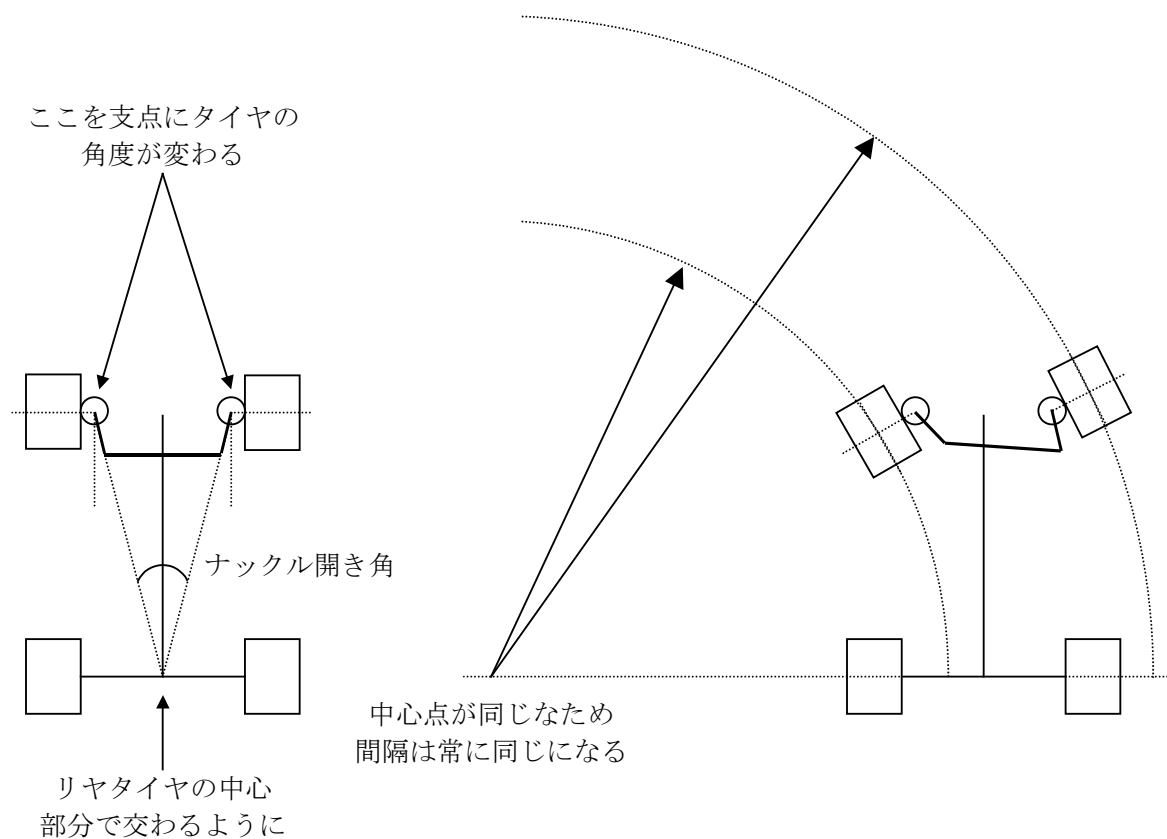
アッカーマン方式とは、通常の車のようにハンドルを切る方法です。左タイヤ、右タイヤの切れ角は実は同じではありません。もし同じ切れ角ならどうなるのでしょうか。



ナックルアームと呼ばれる部分をタイヤと平行に取り付けると、ハンドルを切ったとき、内輪と外輪の切れ角が同じになり、軌跡を見ると交差してしまいます。タイヤの幅は常に一定のため、タイヤに無理がかかります。

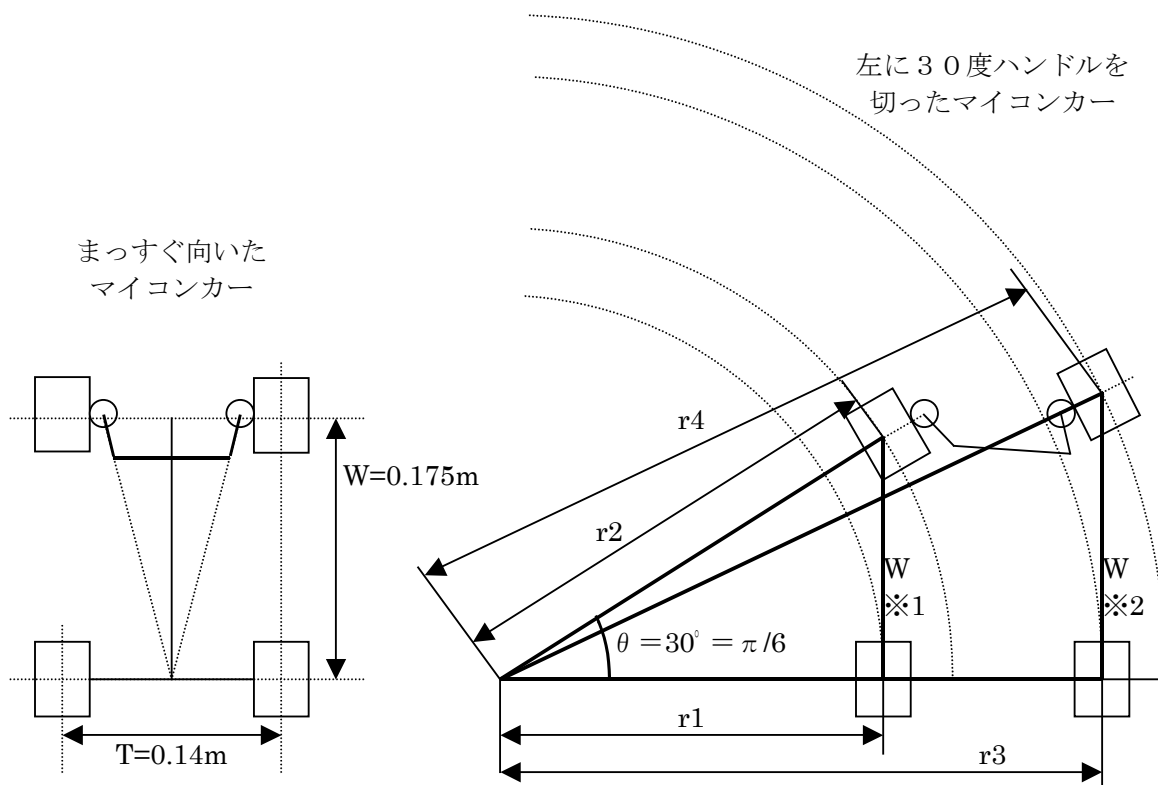
この問題を解決したのが、ドイツ人のアッカーマン、及びフランス人のジャントで、この機構をアッカーマン・ジャント方式、または単にアッカーマン方式と呼びます。

8. 4輪の回転数計算



タイヤに角度を与える左右のナックルアームに開き角を付けていれば、サーボによりタイロッドが左右に動くとナックルアームの動きに差が出て、コーナ内側のタイヤが大きな角度になります。

ナックルアームの開き角度は、リアタイヤの中心部分で交わるようにします。ホイールベース、トレッドにより変わってくるので、マイコンカーに合わせて角度を決める必要があります。



T=トレッド…左右輪の中心線の距離 (キットは 0.17[m]です)

W=ホイールベース…前輪と後輪の間隔 (キットは 0.17[m]です)

角度 θ は、前輪内側タイヤの切れ角です。

※1…実際はホイールベースより短いですが、ほとんど変わらないので W とします。

※2…実際はホイールベースより長いですが、ほとんど変わらないので W とします。

図のように、後輪部の底辺 $r1$ 、高さ W 、角度 θ の三角形の関係は次のようです。

$$\tan \theta = W / r1$$

角度 θ 、 W が分かっていますので、後輪内輪 $r1$ が分かります。

$$r1 = W / \tan \theta = 0.175 / \tan(\pi/6) = 0.303[\text{m}]$$

後輪外輪の半径は、

$$r3 = r1 + T = 0.303 + 0.14 = 0.443[\text{m}]$$

また、前輪内径 $r2$ 、高さ W 、角度 θ の三角形の関係は次のようです。

$$\sin \theta = W / r2$$

角度 θ 、 W が分かっていますので、前輪内輪 $r2$ が分かります。

$$r2 = W / \sin \theta = 0.175 / \sin(\pi/6) = 0.350[\text{m}]$$

8. 4 輪の回転数計算

前輪外輪の半径 r_4 は、底辺と高さが分かっているので、ピタゴラスの定理より、

$$r_4 = \sqrt{(r_3^2 + W^2)} = \sqrt{(0.443^2 + 0.175^2)} = 0.476[\text{m}]$$

一番回転する r_4 を 100 としたときのそれぞれの回転数は、

$$\begin{aligned} r_1 : r_2 : r_3 : r_4 \\ &= 0.303 : 0.350 : 0.443 : 0.476 \\ &= 0.303 \times 100 / 0.476 : 0.350 \times 100 / 0.476 : 0.443 \times 100 / 0.476 : 0.476 \times 100 / 0.476 \\ &= 64 : 74 : 93 : 100 \end{aligned}$$

ハンドル角度 30 度、前輪外輪が 100% で回転するとき、後輪外輪は 93 回転、前輪内輪は 74 回転、後輪内輪は 64 回転することになります。

9. 自作サーボモータの角度指定

今までのステアリング制御は、センサ基板が常にコースの中心にくるような制御をしていました。ラジコンサーボのように、右に何度曲げたい、左に何度曲げたいというように制御したい場合、どうすればよいのでしょうか。

9.1 PD 制御

アナログセンサを PD 制御をしたとき、アナログセンサの値が 0 になるようにサーボモータを制御していました。これを、角度(ボリュームの A/D 値)にすれば良いだけです。

	アナログセンサの値にするとき	角度の値にするとき
比例制御	制御量 $P = k_p \times p$ k_p = 定数 p = 現在のアナログセンサ値 - 目標のアナログセンサ値 = <code>getAnalogSensor () - 0</code> = <code>getAnalogSensor ()</code>	制御量 $P = k_p \times p$ k_p = 定数 p = 現在の角度 - 目標の角度 = <code>getServoAngle () - iSetAngle</code> ※目標の角度を <code>iSetAngle</code> 変数に入れます
微分制御	制御量 $D = k_d \times d$ k_d = 定数 d = 過去のアナログセンサ値 - 現在のアナログセンサ値 = <code>iSensorBefore - getAnalogSensor ()</code>	制御量 $D = k_d \times d$ k_d = 定数 d = 過去の角度 - 現在の角度 = <code>iAngleBefore2 - getServoAngle ()</code>

9.2 プログラム

9.2.1 グローバル変数の追加

グローバル変数を追加します。

/* サーボ関係2 */			
int	iSetAngle;	/* 設定したい角度 (AD値)	*/
int	iAngleBefore2;	/* 前回の角度保存	*/
int	iServoPwm2;	/* サーボ PWM値	*/

9. 自作サーボモータの角度指定

9.2.2 関数の追加

サーボモータの角度指定用の servoControl2 関数を追加します。関数を追加したので、プロトタイプ宣言もしておきましょう。

サンプルプログラムは比例定数 20、微分定数 100、計算後の調整値は 1/2 にしています。この値は、サーボ機構の作り方により違ってきますので各自調整してください。

```

/*****
/* モジュール名 servoControl2
/* 処理概要 サーボモータ制御 角度指定用
/* 引数 なし
/* 戻り値 グローバル変数 iOutPwm2 に代入
*****/
void servoControl2( void )
{
    int i, j, iRet, iP, iD;

    i = iSetAngle; /* 設定したい角度 */
    j = getServoAngle(); /* 現在の角度 */

    /* サーボモータ用PWM値計算 */
    iP = 20 * (j - i); /* 比例 */
    iD = 100 * (iAngleBefore2 - j); /* 微分 */
    iRet = iP - iD;
    iRet /= 2;

    if( iRet > 50 ) iRet = 50; /* マイコンカーが安定したら */
    if( iRet < -50 ) iRet = -50; /* 上限を90くらいにしてください */
    iServoPwm2 = iRet;

    iAngleBefore2 = j;
}

```

9.2.3 割り込みプログラムの追加

割り込みプログラムに servoControl2 関数を追加して、1ms ごとに実行するようにします。

```

/*****
/* タイマ RB 割り込み処理
*****/
#pragma interrupt /B intTRB(vect=24)
void intTRB( void )
{
    unsigned int i;
    asm(" fset I "); /* タイマ RB 以上の割り込み許可 */
    cnt1++;

    /* サーボモータ制御 */
    servoControl();
    servoControl2(); 追加

    /* ブザー処理 */
    beepProcessS();
以下略

```

9.2.4 使い方

main 関数内で使用するときは、

- iSetAngle 変数に、ステアリングモータで角度を指定したい A/D 値を代入します。
- プログラムは、「servoPwmOut(iServoPwm2);」を実行します。「servoPwmOut(iServoPwm);」とすると、センサ基板がコースの中心になるようなステアリング制御になります。2 が付くか付かないかの違いです。

下記に、main プログラムの一番最初で角度指定した例を示します。このプログラムできちんと角度指定できているか iSetAngle 変数の値を変えて実験してみましょう。

なお、元々のプログラムの iAngle0 変数の設定は走行開始直前なので、iAngle0 の設定をいちばん最初にしています。

```

/*****
/* メインプログラム
/*****
void main( void )
{
    int i;

    /* マイコン機能の初期化 */
    init();                /* 初期化                */
    asm(" fset I ");      /* 全体の割り込み許可  */
    initBeepS();          /* ブザー関連処理      */

    /* マイコンカーの状態初期化 */
    motor_mode_f( BRAKE, BRAKE );
    motor_mode_r( BRAKE, BRAKE );
    motor_f( 0, 0 );
    motor_r( 0, 0 );
    servoPwmOut( 0 );
    setBeepPatternS( 0x8000 );

    cnt1 = 0;
    while( cnt1 <= 10 );
    iAngle0 = getServoAngle(); /* 0度の位置記憶      */

    iSetAngle = 100;
    while( 1 ) {
        servoPwmOut( iServoPwm2 );
    }
}

```

以下略

角度指定ができたことを確認できたら、実際の走行プログラムに組み込んでマイコンカー制御に使用します。下記プログラムは新しくパターン52を作り、A/D 値が-50になるような位置に自作サーボモータを移動させる例です。

```

case 52:
    iSetAngle = -50;
    servoPWM( iServoPwm2 );
    break;

```

10. 参考文献

- ルネサス エレクトロニクス(株)
R8C/38C グループ ユーザーズマニュアル ハードウェア編 Rev.1.10
- ルネサス エレクトロニクス(株)
M16C シリーズ,R8C ファミリー用 C/C++コンパイラパッケージ V.6.00
C/C++コンパイラユーザーズマニュアル Rev.1.00
- ルネサス エレクトロニクス(株)
High-performance Embedded Workshop V.4.09 ユーザーズマニュアル Rev.1.00
- ルネサス半導体トレーニングセンター C言語入門コーステキスト 第1版
- 電波新聞社 マイコン入門講座 大須賀威彦著 第1版
- ソフトバンク(株) 新C言語入門シニア編 林晴比古著 初版
- 共立出版(株) プログラマのための ANSI C 全書 L.Ammeraal 著
吉田敬一・竹内淑子・吉田恵美子訳 初版

マイコンカーラリーについての詳しい情報は、マイコンカーラリー公式ホームページをご覧ください。

<http://www.mcr.gr.jp/>

R8C マイコンについての詳しい情報は、ルネサス エレクトロニクス(株)のホームページをご覧ください。

<http://japan.renesas.com/>

の製品情報にある「マイコン」→「R8C」でご覧頂けます